Analysis of the Security of Split Manufacturing of Integrated Circuits

by

Jonathon C. Magaña

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2018

Date of final oral examination: 08/20/2018

The dissertation is approved by the following members of the Final Oral Committee:

Azadeh Davoodi, Associate Professor, Electrical and Computer Engineering

Yu Hen Hu, Professor, Electrical and Computer Engineering

Younghyun Kim, Assistant Professor, Electrical and Computer Engineering

Dan Negrut, Professor, Mechanical Engineering

Dedicated to my son, Julian X. Magaña

ACKNOWLEDGMENTS

I would like to thank to my advisor, Professor Azadeh Davoodi, for her guidance and the opportunity work with her.

I would also like to thank my mother, Mary Flynn, for keeping me on track and for her invaluable perspective and suggestions.

Finally, I would like to thank my husband, Sergio Magaña, for his support throughout the entire process. I'm the luckiest badger to have a guy like you!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

Split manufacturing is a technique that allows manufacturing the transistor-level and lower metal layers of an IC at a high-end, untrusted foundry, while manufacturing only the higher metal layers at a smaller, trusted foundry. Using split manufacturing is only viable if the untrusted foundry cannot reverse engineer the higher metal layer connections (and thus the overall IC design) from the lower layers.

This work begins with a study of the effectiveness of proximity attack as a key step to reverse engineer a design at the untrusted foundry. We propose and study different proximity attacks based on how a set of candidates are defined for each broken connection. The attacks use both placement and routing information along with factors which capture the router's behavior such as per-layer routing congestion. Our studies are based on designs having millions of nets routed across 9 metal layers and significant layer-by-layer wire size variation. Our results show that a common, Hamming Distance-based proximity attack seldom achieves a match rate over 5%. But our proposed attack yields a relatively-small list of candidates which often contains the correct match. We also propose a procedure to artificially insert routing blockages in a design at a desired split level, without causing any area overhead, in order to trick the router to make proximity-based reverse engineering significantly more challenging.

This work goes on to analyze the security of split manufacturing using machine learning. We consider many types of layout features for machine learning including those obtained from placement, routing, and cell sizes. For the top split layer, we demonstrate dramatically better results in proximity attack compared to the same

proximity attack without machine learning. We analyze the ranking of features used by machine learning and show the importance of how features vary when moving to the lower layers. Since the runtime of our basic machine learning becomes prohibitively large for lower layers, we propose novel techniques to make it scalable with little sacrifice in effectiveness of the attack.

This work concludes by proposing a routing-based defense mechanism against the proximity attack. Our model is able to simultaneously generate multiple routing solutions which exhibit a trade off between the enhanced security versus degradation in wirelength. This means the generated solutions are Pareto-optimal with respect to each other such that for example, for a desired security, one solution is always best in terms of wirelength overhead, or for a desired allowable wirelength overhead, one solution always has the highest security. Our algorithm guarantees its process does not generate overflow of routing resources and gives flexibility to the designer to eventually select one of these generated solutions based on a desired level of security or tolerable wirelength overhead. This addresses an issue common in prior work, where a designer lacks control over a routing obfuscation technique's wirelength overhead. The effectiveness of our algorithm is demonstrated with experiments using the ISPD'11 benchmarks featuring up to two million nets.

Chapter 1

Introduction & Motivation for Split Manufacturing

1.1 Introduction

Split manufacturing is a fabrication model of Integrated Circuits (ICs) which enables secure use of a high-end, untrusted foundry located off-shore. In this model, only partial information about the chip is sent to the untrusted foundry. Therefore the chip is only partially fabricated [18, 29]. For a layout described as a network of connected transistors, the partial information corresponds to the complex steps of fabricating the transistor layer and a small subset of metal layers that are immediately above it such as M1 and M2 that include the pins of standard cells (i.e., Front End of Line, FEOL).

Fabrication of the rest of the connections, captured by the higher metal layers (i.e., Back End of Line, BEOL), typically won't require a complex fabrication process and can be done by a smaller, trusted company, on-shore. This is because the wires of BEOL can be up to 10X wider so they are significantly fewer in number and have more spacing between them as shown in Figure 1.1(a) adapted from [2]. Fewer wires on the top layers simplifies BEOL mask generation and makes it easier for the trusted party to align FEOL with BEOL [15].

Split manufacturing is an attractive fabrication model for ICs utilized in military applications because it decreases the likelihood of reverse engineering the design and inserting targeted hardware Trojans on the chip. It is also an attractive option for small companies that are interested in using a high-end fabrication facility but

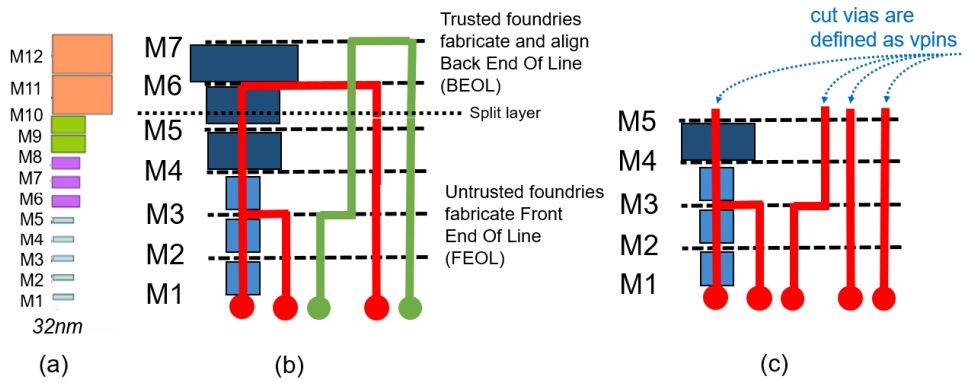

Figure 1.1: (a) Wire size variations in 32nm; (b) Entire design showing split layer; (c) FEOL showing vpins. Figure adapted from [2].

do not want to give away their "secret sauce", in case of concern that the untrusted foundry may attempt to steal the Intellectual Property [1, 3, 4, 5, 6, 13, 23, 37, 38, 39, 40, 41, 49, 51].

Given the layout of an arbitrary design, deciding on a split level may pose a tradeoff between the degree of security at the untrusted foundry versus ease of fabrication at the trusted foundry. On one hand, it is desired to split the chip at a lower layer; this means a higher number of incomplete connections because the lower layers have thinner wire sizes and support more routing tracks. On the other hand, it is desired to split at a higher layer because of easier fabrication and fewer alignment issues to add BEOL to the FEOL [15].

1.2 Preliminaries

As a key step in split manufacturing, a design layout generated with a router is split at a particular metal layer to allow for fabrication of FEOL and BEOL at different facilities. Routing of a design consists of the two steps of global and detailed routing. In this work we target the global routing stage to build our defense mechanism against split manufacturing attacks.

Global routing involves determining the wiring connections necessary to connect the cells of an integrated circuit (IC) based on placement information, the set of all nets (netlist), and technology specific design rules. The routing is done according to a global grid which is coarser than the final, detailed routing grid, and each bin in the grid may contain several standard cells. The placement information consists of the location of individual cells on the transistor layer of the IC. Each individual net in the netlist is defined by the pin locations of a specific subset of cells that need to be connected during global routing. Moreover, during the global routing process, design constraints such as capacity of *routing* resources must be taken into account.

Minimizing the total wirelength is one of the most important optimization objectives of the global routing process. **Wirelength (**WL**)** is defined as the number of edges along which the wire connection travels, where the edges belong to the grid-graph associated with the global routing grid. Each edge in the grid graph has a capacity which specifies the maximum number of wires that can be crossing that edge without exceeding its capacity. Otherwise, exceeding an edge's capacity with too many wire connections is defined as **overflow**. Indeed, another primary objective of global routing is to achieve an overflow-free routing solution, and otherwise minimize overflow as much as possible.

The global routing grid-graph is a 3D graph with "horizontal" edges representing each metal layer, and "vertical" edges representing the vias that connect different layers to each other. Vertical edges are the inter-layer via connections between a point on one layer of the grid graph up to the point on the layer above it.

In this work, we consider not only horizontal edge capacity but also vertical edge capacity. For simplicity, inter layer vias are often assumed to have infinite capacity [16]. However this assumption would be too simplistic for split manufacturing. In this work, we chose to include a more realistic, finite, via capacity model

to better consider overflow in the vertical edges. We define this vertical edge capacity as the square of the edge capacity at the lower layer of the interlayer via. In this way we are calculating that the rectangular area that comprises the footprint of available via capacity up to the next layer can be calculated as the capacity of an edge along that rectangular area squared. [16, 31, 32, 33]. We did this because any routing-base defense mechanism naturally favors routing nets using higher layers and thus makes expanded use of vertical resources; we considered it an oversimplification to assume infinite capacity of these vertical edges.

Solutions generated with our algorithm are guaranteed to have zero overflow because the input of our algorithm is an already overflow-free solution, and our algorithm is able to improve this solution to enhance its security (and trade it with wirelength) without causing any overflow.

1.3 Attack Model

In the most commonly used model of split manufacturing, an untrusted foundry has received a layout file which contains information about the transistor layer and some metal layers that are immediately above it (FEOL). The attack model is one in which the goal of the untrusted party is to guess the missing connections in the higher layers (BEOL) using the FEOL information. The layout file which is typically described in the GDSII format allows quick generation of a gate-level description of the partially-connected network. It includes information on where each cell is placed, as well as the layout of each cell which includes factors such as cell type, electrical properties, drive strength, and pin shape. The route fragments of the partial network are also completely specified on each layer corresponding to the FEOL. The attack model considered in this dissertation does not include inserting hardware Trojan or reverse engineering the design.

Moreover, there are other variations of split manufacturing based on 2.5D or 3D integration in which the design is partitioned into multiple dies which are fabricated by either trusted or untrusted parties, and then get integrated into one package [17]. We note techniques for 2.5/3D integration will be very different than the ones studied in our attack model because in our model, for an already-routed design, the nets which should get broken are forcibly decided by the split level. However in 2.5D/3D integration there is flexibility to decide how a design should be partitioned.

In this work, we use **vpin** to refer to a Virtual Pin or the point where a net is broken at a split layer, as shown on Figure 1.1(c). Where Figure 1.1(b) shows the routing of a design featuring two nets, Figure 1.1(c) shows this from the point of view of an untrusted foundry for the same nets when the split level is at layer M5. The inter-layer vias are drawn to reflect the increase in wire sizes in the higher layers. When a net is cut at the split layer, the resulting vpins associated with that net are called **matching vpins** because they must be reconnected during BEOL fabrication. Unlike pins of the cells which are typically located on layers 1 or 2, the vpins are located on the split layer which may be on layer 3 or higher.

Chapter 2

Related Work & Contributions of this work

2.1 Related Work

Studies that provided an earlier foundation for split manufacturing concluded that it is inherently insecure. These studies asserted that, since their analysis indicated an attacker would be able to correctly infer most of the missing connections which lead to the overall design, design obfuscation is necessary [29, 45]. It is important to note that these early studies focused on smaller designs that don't reflect modern IC design complexity.

The conclusions in one of the relevant early studies were drawn assuming that nets are broken at the location of the I/O ports [29]. That work identified the I/O port locations using a circuit partitioning tool, which minimizes the number of connections between partitions. This approach is unrealistic since it is unrelated to how a design is actually routed. The study in [29] also did not take into account wire size variations.

Another earlier study, recently updated, used a network flow-based attack to guess missing connections in a circuit [44, 45]. This study considered factors including proximity, the rarity of combinatorial loops, load capacitance constants, etc. to improve the number of correctly-guessed connections. Their attack was shown to be successful on the benchmarks they considered [7, 14].

More recent studies have focused on attacking larger designs, as in [28] with techniques such as the network flow based attack in [44], and machine learning, as

described in Chapter 4 and in [50]. These more recent studies have also concluded that obfuscation is needed to protect a design fabricated with split manufacturing.

A number of prior works have also focused on developing defense techniques to enhance the security of split manufacturing. In [30], the authors proposed two defense techniques: gate-level graph coloring and same-type gate clustering. Both techniques rely on placement level obfuscation that randomizes cell placement to reduce proximity-based vulnerabilities. They define an information-based metric to measure the effectiveness of their defenses, which incur area, power and delay overhead. Their information-based metric quantifies how much information an attacker can derive from the placement level layout data [30]. The work concluded that their defense is more effective at lower split layers, and improved security when compared to the placement obfuscation defense in [45].

Another work introduces a netlist randomization-based defense that results in obfuscation of both placement and routing information [27]. Custom cells are used to help handle wire detours and facilitate restoring the correct design functionality in the BEOL. The work reported that their approach improved security by increasing the number of vias (vpins) at the split level and reducing the effectiveness of the network flow based attack outlined in [45].

Several recent works focused on the concept of wire lifting, which has the main goal of routing additional nets above the split layer [10, 28, 46]. These works used a variety of techniques to achieve this goal including wire extension based on a wire congestion density analysis as in [10], prioritizing nets that have a higher fault observability for lifting as in [46], and identifying both high fanout and short wirelength nets for lifting to achieve higher flexibility in routing and to obfuscate the most easily attacked nets respectively in [28].

These works faced some common challenges, the most important being that wire lifting inevitably results in increasing the overall wirelength of a design. In [10], the authors also sought to address issues of manufacturability which they

maintain are often neglected in split manufacturing research. Finally, in [28] the authors discussed the difficulty in selecting a split level that balances security and cost for their approach because splitting at a lower level increases security but is more costly for BEOL manufacturing.

Table I shows a summary of related work that explores defenses against reverse engineering attacks in split manufacturing. The related works consider a variety of approaches to attacking split manufacturing.

Each entry in Table I shows:

- The **benchmark** or benchmarks chosen for analysis [7, 14, 42].
- The main **security metric(s)** discussed in the related work, which relates to the **defense overhead** column.
- The main **defense technique(s)** proposed by the related work.
- The **defense overhead** shows the overhead of the defense technique used to protect against the attack.

The **security metric** column specifies the way each related work measured the efficacy of an attack. Hamming Distance indicates that the work calculated the distance between a cut wire (vpin) and its actual match as a measure of security. If the Hamming Distance increases because the vpin was moved further away by employing the specified defense technique, a proximity attack would be more difficult. Another metric used is number (#) of known functions. In this attack model, the attacker is trying to determine the specific high-level functional block that each cut connection was a part of in the original design. The more functions known to an attacker, the more successful the attack. For k-isomorphism, a connection must be indistinguishable from at least k-1 other connections. The "information-base" metric measures how much is known about a cut connection based on its location. Fault observability characterizes how easily a fault can be detected and is used to identify

Source	Benchmark(s)	Security metric	Defense Technique	Defense Overhead
[29]	ISCAS'85	Hamming distance, # correct connections	pin swapping	additional wirelength
[18]	standard cell test cases	# of known functions	placement obfuscation, dummy cells	77% area
[39, 40]	test cases	# of wrong connections	placement obfuscation, dummy transistors	stated to be negligible
[17]	ISCAS'85, DES circuit	k-isomorphism	wire lifting	200% area and delay
[48]	ISCAS'85, ITC-99	Hamming distance	placement obfuscation	14.27% wirelength, 5.29 % area
[44, 45]	ISCAS'85, ITC-99	# correct connections	placement obfuscation	3.3% wirelength, 0.27% delay
[30]	MCNC and ISCAS'85	information based	placement obfuscation	50% power, 18% delay, 60% area
[27]	superblue, ISCAS'85	hamming distance, %recovered nets, # vpins	routing and placement obfuscation	3.5% power, 2.7% delay
[28]	superblue, MCNC ISCAS'85, ITC-99	correct connection rate, %recovered nets	wire lifting	9.2% area, 10.7% power, 15% delay
[46]	ISCAS'85, ITC-99	fault observability, # wrong connections	wire lifting, dummy pins	2.9% wirelength, 0.23% delay
[10]	ISCAS'85, ITC-99	hamming distance, # wrong connections	wire lifting, dummy wires	3.68% wirelength, 0.83% delay

Table I: Related work on split manufacturing

nets that are good candidates for obfuscation. Lastly, number (#) of correct/wrong connections measures how many correct/incorrect connections are made during an attack. As the number of wrong connections increases, security is increased because the attack was less successful. The (%) recovered nets metric is similar to the (#) correct connections.

The **defense technique** lists the work's proposed defense. Defenses include various forms of placement obfuscation, such as insertion of dummy pins, cells and/or, and pin swapping. Defenses also consist of forms of routing obfuscation, such as insertion of dummy wires, and wire lifting where additional nets are routed above the split layer.

The **defense overhead** lists the source of the overhead (such as increased delay, area, routing time, or wirelength) and also lists the approximate amount of increased overhead as a percentage of the original value of the indicated source of overhead.

Other related works have focused on hardware Trojan insertion prevention. In [34] the authors proposed built-in self-authentication technique where obfuscation is used to prevent hardware trojan insertion at the FEOL; as an added benefit, the authors indicate this method increases the overall security of the FEOL information by making it more difficult to reverse engineer.

In [20], which is a follow up to [17], the authors proposed an integer linear programming (ILP) approach to wire lifting, which also considered dummy cell insertion to achieve obfuscation. Since the goal of the attacker in this work is to insert an undetected hardware trojan during FEOL fabrication, it is not directly comparable to other works where the goal of the attacker is to reverse engineer the BEOL design information.

Yet another related work considered the attack model where both FEOL and BEOL are untrusted [43]. This work considered machine learning and geometric pattern matching based attacks to evaluate the security of split manufacturing where both foundries are untrusted. The authors proposed a cell-based obfuscation defense to protect against this attack model.

In this work we do not consider the possibility of hardware Trojan insertion in our analysis.

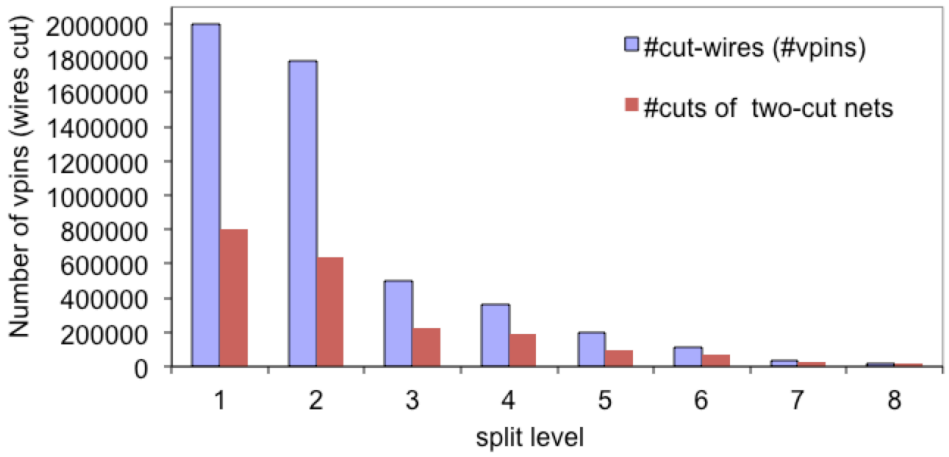

Figure 2.1: Number of vpins (i.e., wires cut) per split level

2.2 Motivation & Open Challenges

Split manufacturing presents challenges but is an attractive option for manufacturing ICs. Among the challenges is the complexity of modern circuit designs. For complex designs, there can be hundreds of thousands of nets that are routed up into the higher metal layers of a circuit. This results in a correspondingly large number of cut wires when a design is split horizontally at the split level.

Figure 2.1 shows the number of cut-wires as a function of split level for benchmark `superblue1` from the ISPD'11 suite. This benchmark has 9 metal layers creating 8 split levels; for example split level 4 is when the split is between M4 and M5. At each split level, a cut-wire creates a vpin at the 3D point where the wire is cut. In a complete design a via is placed at a vpin location to connect it to the higher layers.

At each split level, we report the total number of vpins in blue as well as the number of vpins for two-cut nets in red. Two-cut nets are the ones which are broken at only two points at the split level. This is the minimum number of cuts *if* a connection is broken and it is possible that number of vpins of a broken net is more

than 2. In general, the number of vpins is an upper bound on the number of broken nets.

As can be seen in Figure 2.1, the number of vpins in both cases are extremely high for this benchmark, for example almost 2M for the blue case for split level 1. As can be seen, the numbers significantly drop when moving to the higher split levels. This is expected because the wires are significantly wider on the top layers, so fewer routes can pass (thus get cut) in those layers. But even at the highest split level, i.e., level 8, we have over 16000 vpins for the blue case. The number of vpins are still high for the red case and ranges from 0.8M at level 1 down to 13000 vpins for level 9.

Figure 2.1 essentially hints at the difficulty of this reverse engineering problem, merely due to the significantly large size of today's designs. Even given this challenge, other studies have addressed and analyzed split manufacturing and concluded that the process is insecure due to the ability of an attacker to recover the overall design by guessing the majority of missing connections. Initial studies had a simplistic model of routing and used benchmarks that didn't reflect modern IC designs [29]. However, more recent studies have used a range of benchmarks and a more representative model of routing and also concluded, like prior studies, that design obfuscation is essential [27, 50].

As discussed in Section 2.1, both early studies and more recent work have highlighted how the security of split manufacturing can be vulnerable to proximity attacks [28, 29, 45]. In order to ensure the security of split manufacturing, there are several open challenges that are apparent when considering related work that must be addressed.

One open challenge is how to assess the vulnerability of a design to proximity-based reverse engineering attacks.

Another open challenge is that early studies did not use benchmarks large enough to reflect the reality of modern IC design and used unrealistic assumptions about how a design is split for manufacturing.

Finally, the conclusion of recent studies and this dissertation is that obfuscation is necessary to protect a design fabricated with split manufacturing. An open challenge with obfuscation based defenses is that routing obfuscation leads to an increase in the overall wirelength of a routed design.

2.3 Contributions

First in Chapter 3, we address the challenge of using small benchmarks to analyze the attainable security of the split manufacturing process. We use the recent, larger ISPD 2011 routability-driven placement benchmarks which contain designs having millions of nets routed across 9 metal layers with significant layer-by-layer wire size variation [42]. These benchmarks more realistically reflect modern IC designs. More recent studies have also adopted larger benchmarks [27, 28].

The challenge of assessing the vulnerability of a design to proximity attacks remains an important issue. In order to make this assessment, in Chapter 3, we introduce and study novel ways that a close neighborhood can be defined to identify promising candidate vpins (cut wires from nearby nets that are likely the matching connection). Unlike earlier studies, we explore proximity attacks on a design that is split on a variety of metal layers based on both placement information and routing information including congestion.

Our results show that defining proximity based on picking the closest vpin using Hamming Distance only achieves a match rate of at most 5%. However, we show there is a significantly higher chance that our proximity attack (and specifically the congestion-driven one), identifies a relatively-small list of candidates that includes the matching vpin. These observations indicate that proximity alone is in no way

sufficient but can be a good starting point to significantly narrow down the list of candidates.

Moreover, in Chapter 3, we also propose an novel obfuscation technique to address the open challenge that designs must be obfuscated to protect against reverse engineering attacks. Although recent studies have focused on obfuscating circuit design or scrambling cut pin locations at the split level to make proximity and other attacks more challenging [18, 26, 29, 45, 47], our technique uses a different approach which is compatible with and would not require altering any of the existing CAD tools. It essentially makes the router route more nets at the higher metal layers, thus creating a higher number of broken connections. We show in our experiments that the proximity attacks become significantly more challenging after blockage insertion. Our procedure is designed to ensure the design remains routable just as it was prior to blockage insertion.

Next in Chapter 4 we use machine learning to further improve our proximity attack in the first work. With an identical experimental setup as Chapter 3 and [24, 25], we show significant improvement in the effectiveness of the attack over the results reported in Chapter 3. For this analysis we include additional placement information and layout features and ensure separation of training and testing data sets. The goal of this analysis is similar to the previous chapter: to find a small list of candidates for each broken net, where the list of candidates ideally contains the actual match. [1]

In our experiments in Chapter 4 we show that, for example for split layer 8, the size of the list of candidates for the broken nets is on-average 8% of the one in Chapter 3 while the likelihood of including the actual match in the list is 99.9% compared to 42.72%. Additionally, our improvements to the scalability of machine

[1] The work in Chapter 4 was done in collaboration with Boyu Zhang who conducted the machine learning modeling aspects of the research.

learning reduced the average attack runtime on layer 6 from 21.65 hours to 10.73 hours.

After showing in chapter 4 that higher layers can be successfully attacked using machine learning, in Chapter 5 we propose a new routing algorithm as a defense mechanism. This also addresses the open challenge that design obfuscation is needed in order to protect a design against attacks by the untrusted foundry. The algorithm in Chapter 5 is novel because, unlike prior work, it simultaneously generates multiple routing solutions which provide a tradeoff in the wirelength versus security space. This means the generated solutions are Pareto-optimal with respect to each other such that for example, for a desired security, one solution is always best in terms of wirelength overhead, or for a desired allowable wirelength overhead, one solution always has the highest security. We use a Pareto-algebraic approach similar to [35], but instead tailored to explore the tradeoff between wirelength and security. Generation of multiple solutions is done in a flexible manner and can be controlled by the designer.

In our experiments in Chapter 5 we show the effectiveness of our algorithm using the same ISPD'11 benchmarks we used in Chapter 3 [42]. We observe a gradual improvement in security in exchange for degradation in wirelength, where on average our security metric increases by 0.46 per unit increase of wirelength. We also observe that the percentage increase of our security metric is very closely related to the percentage increase in the small list of candidates that a proximity attack identifies for each cut net. For example, an increase in security by 21.15%, 67.66%, and 91.72% results in an increase of average candidate list size of 24.39%, 62.04%, and 97.16% respectively.

A summary of the contributions in this work is listed below:

- We conduct studies using the large ISPD'11 benchmarks compared to the smaller benchmarks used by early prior work [42].

- We propose and analyze intuitive ways to identify a small set of candidates for each broken connection based on physical proximity, utilizing place & route information as well as router behavior in creating congestion. We then propose an automated procedure to artificially alter the netlist, and yet use the existing CAD tools, and show this can significantly complicate the proximity attacks.

- We further improve our analysis by using machine learning to enhance our proximity attack. We incorporate more layout features than before including placement, routing, cell sizes, and cell pin types. We propose novel techniques for effective realization of training and testing in machine learning. Since the runtime of our basic machine learning becomes prohibitively large for lower layers, we propose novel ways to make it scalable, achieving significant runtime improvement without much sacrifice in the effectiveness of the attack. We then study ranking of the features with each rank associated with a weight signifying its importance. Using the ranked features, we study relative importance of the features across the layers. We show routing features are the most important ones, and as we move to lower layers more features become important. This is the first work to use machine learning to study the security of split manufacturing.

- The final contribution of this dissertation is a new algorithm that can simultaneously generate multiple routing solutions which are Pareto-optimal relative to each other in the wirelength versus security tradeoff space. To aid in our analysis, we define a new security metric to measure the effectiveness of routing obfuscation. Further, our algorithm has the flexibility to allow the designer to control the number of generated solutions, each of which maximizes the security metric for an acceptable amount of increased wirelength or minimizes wirelength for the desired value of the security metric. Giving the

designer control over wirelength overhead in this manner is an improvement over the lack of control of overhead in prior work routing obfuscation techniques. This is a novel approach to obfuscating a design to increase security during the split manufacturing process that has not been attempted in other works.

Chapter 3

Initial Analysis and Results on Proximity Attacks

In this chapter, we study how effectively an untrusted foundry can guess the broken connections in a design that has been split horizontally to allow for fabrication of the FEOL lower layers off shore. We use both placement and routing information to define our proximity attacks. We define a small neighborhood around a broken connection, designated as a vpin, that contains other broken nets that are likely the match for the broken connection we are considering. Based on our observations, proximity can be a good starting point to significantly narrow down the list of candidates, but alone is in no way sufficient to identify matches for all vpins. To obfuscate the proximity information, we then propose a procedure to insert artificial routing blockages at a desired split level to trick the routing tool into creating more vpins at the split level.

We do not deal with power constraints or directionality, or timing constraints in our analysis, instead analyzing a solely proximity based attack without these additional filters that may further increase the effectiveness of the attack.

In the remainder of this chapter, in Section 3.1 we present and analyze our proposed proximity attacks with additional discussion for nets which are cut in more than two places at the split level. Our blockage insertion procedure is discussed in Section 3.2 with extensions for timing-critical nets and discussion of branch insertion technique. Our simulation results are presented in Section 3.3, followed by conclusions.

3.1 Proposed Proximity Attacks

3.1.1 Overview and Goals

Given the large number of vpins (cut-wires) at any of the split levels, a practical attack strategy is that for each vpin, a subset of other vpins within its close physical proximity are considered as its matching candidates. In this section we propose a number of ways to define this neighborhood in order to filter a significantly-smaller set of candidates per vpin.

3.1.2 Methods to Define Proximity

We discuss 4 techniques to identify a search neighborhood for each vpin. What we desire is to define a small neighborhood so the number of candidate vpins which fall in that neighborhood are feasible for further pruning. Our second goal is to increase the likelihood of actually including the matching vpin in the identified neighborhood. Achieving these goals can be a major step in simplifying the attack because, for a small set of candidates, it becomes feasible to check various pruning rules (including formation of combinational loops and many others) in order to guess the matching candidate. Our techniques aim to address these two goals.

Our first technique defines a **placement proximity** from the cells' placement information. Here the location of each vpin is taken from the pin location of the corresponding standard cell that is connected to it. For a vpin, we define the search neighborhood to be a square centered around the corresponding pin with an area equal to the average areas of the bounding boxes (BB) in a typical design. It can also be measured based on BBs of the uncut wires in the design under attack because the number of uncut wires are also very large in practice. The average BB of a routed net (measured by placement pins) is significantly smaller than the total routing grid granularity, thus it defines a significantly smaller search neighborhood. It is also

a good search space intuitively because the BB of a net includes all its pins so the average BB value will include the pins for many nets.

Figure 3.1(a) shows the first 2 routing layers in a sample design. For simplicity the figure shows standard cells and layer M1 on one aggregate layer because typically the majority of M1 is used for connections inside each standard cell.

Figure 3.1(b) shows the FEOL view when the split level is M2. It also shows the candidates identified by placement proximity for the considered pin (or vpin)[1].

Our second proximity method is also shown in Figure 3.1(b) which is referred to as **routing proximity**. We define the proximity as a square area centered around this vpin. We define the area of the square based on the average BBs of the vpins on that layer in the design. Specifically if two or more vpins belong to the same net at a specific metal layer, the BB defined by these vpins is measured and then averaged over all the vpins on the layer. (From the attacker's view, information of a typical (full) design can be used to identify the vpins and compute the BBs at a target split level.)

The above procedure for identifying a search neighborhood using the vpin locations results in different search neighborhood sizes as the split level varies. (However the size of search neighborhood is the same within a split level.) Indeed, we show the search neighborhood becomes larger when moving to the higher layers because the wires become significantly wider, thus the vpins of the same net will be typically further apart from each other. However a larger search area at a higher layer does not necessarily include a higher number of candidate vpins because fewer wires can get routed on the top layers due to the larger (wider) wire sizes.

As shown in Figure 3.1(b), routing proximity may identify a new set of candidates compared to placement proximity. Intuitively, it is a suitable search area

[1]For simplicity, in cases shown in Figure 3.1, the search area(s) are not drawn to be centered around the considered (v)pin.

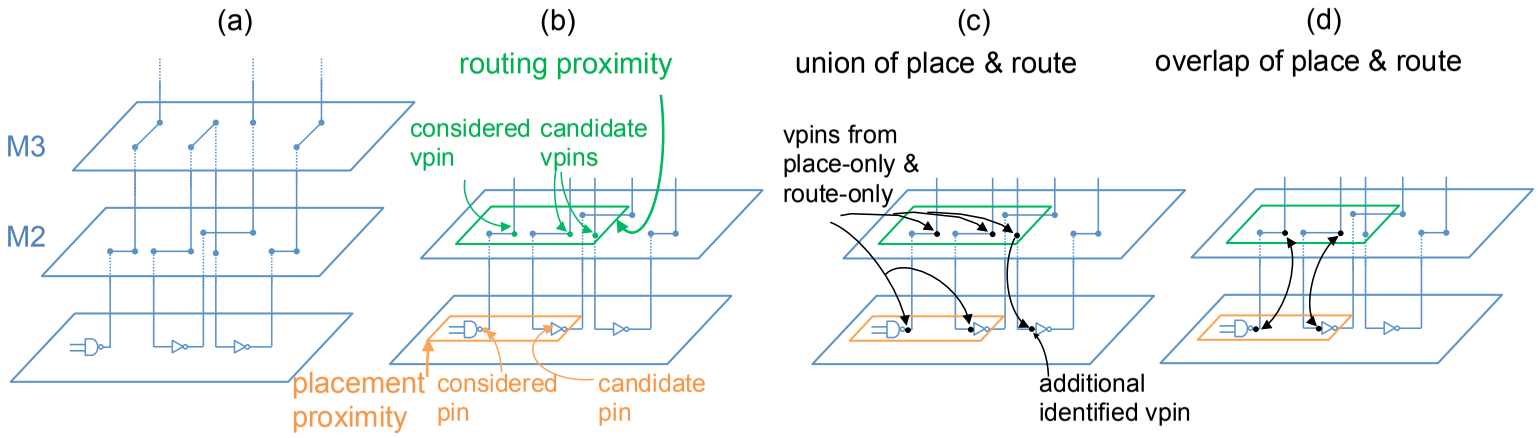

Figure 3.1: Illustration of various proximity definitions.

because by definition, the vpins of the same net all fall within the area identified by the BB of those vpins.

We also consider a variation of routing proximity that takes into account routing congestion. We refer to this as **crouting proximity**. Here the search area is defined based on estimated routing congestion on each split level and varies for each vpin for the same split level. Specifically, we define the search area for each vpin such that the ratio of number of vpins to the search area is equal across all the vpins in the split level. If a vpin is at an area with high routing congestion, then the search area will be expanded (uniformly from all directions centered around the vpin) until the *vpin density* in the new search area reaches a target value or the search area grows to four times its starting value. For a split level, we set this as average vpin count which fall within a BB of vpins of the same net.

Our third proximity technique is shown in Figure 3.1(c). It is the **union of placement and routing proximities**. The candidate list is initially composed of the pins identified by placement proximity. Next, for each vpin found by routing proximity, the corresponding pin is also included in the list. So the number of candidates identified by this technique may be larger than sum of candidates in placement-only and routing-only techniques as seen in the figure.

Our last proximity technique is **overlap of placement and routing proximities**. It includes a subset of pins identified by the placement proximity list which have their corresponding vpins included in the routing proximity list. This is shown in Figure 3.1(d). Obviously the list of candidates will be smaller than both placement-only and routing-only proximities. Intuitively, the overlap identifies a subset of vpins (from a set of considered pins) which *may* be more likely to *point* towards the direction of the matching pin.

In our experiments presented in Section 3.3 we compare our proximity techniques and show some of them can be quite effective to successfully narrow down the list of matching candidates while including the actual match. This shows there

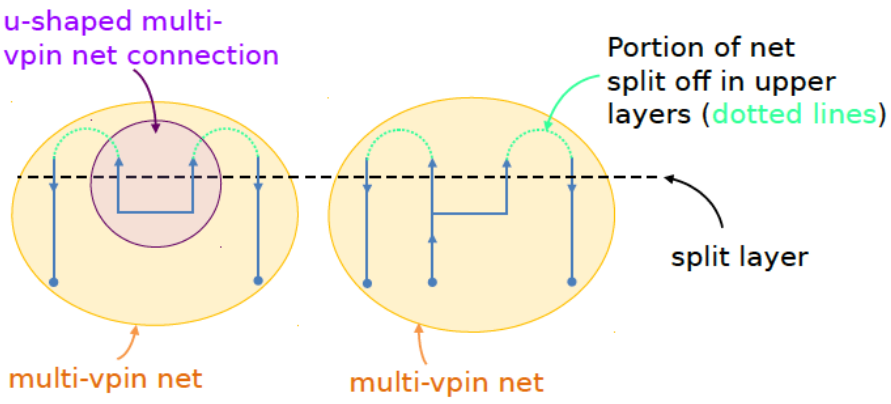

Figure 3.2: Defining multi-vpin nets

are opportunities for aggressive attacks on a much smaller subset of candidates for each vpin. Therefore to address this issue, in the next Section we discuss a novel technique to reduce the effectiveness of proximity attacks, in general, regardless of how proximity is defined.

3.1.3 Handling Multi-Vpin Nets

As a logical next step to the proximity attack on 2-vpin nets, we extend our proximity matching techniques to multi-vpin nets. A multi-vpin net has 3 or more vpins at the split level, so it is a net that is cut in at least 3 places at the split level. In some multi-vpin nets, two or more vpins are connected in a "u-shaped" connection without a connection down to layer one as shown in Figure 3.2. In other cases, two or more vpins connect down to the same pin on layer one, also shown in Figure 3.2.

We expand the proximity algorithm for 2-vpin nets to handle the multi-vpin case. The first step is to identify valid multi-vpin nets, as defined in Figure 3.2. The expanded algorithm for the multi-vpin case uses an expanded search area when considering candidates for a match.

To determine the size of the expanded search area we analyzed the `superblue1` benchmarks from the ISPD-2011 benchmark suite [42] for multi-vpin nets. Table II

reports the average bounding box sizes for multi-vpin nets at various splits levels. After analyzing this data, we chose the multi-vpin bounding box as the 4-vpin net value (for 3-vpin, 4-vpin and 5-vpin nets), and then scaled it up by a factor of 1.5 for 6-vpin and 7-vpin nets, and by factor of 2 for 8 or higher vpin nets. These factors provide an approximation of the bounding box size trends observed in Table II.

Split level	Nets with:	3 vpins	4 vpins	5 vpins	6 vpins	7 vpins	8 vpins	9 vpins	10+ vpins
2	# of nets	36921	37888	16172	10295	5937	4002	2651	10293
	Average BB	61.0	46.6	69.8	67.3	74.6	83.6	92.3	121.6
4	# of nets	3468	8275	1138	2159	342	773	158	1259
	Average BB	142.6	153.3	184.8	225.3	213.7	289.6	274.6	328.0
6	# of nets	188	3223	61	777	27	357	12	407
	Average BB	179.8	221.3	284.6	312.6	384.4	411.6	489.0	497.3
8	# of nets	0	152	1	29	0	4	0	6
	Average BB	n/a	285.9	284.0	416.6	n/a	894.0	n/a	1043.7

Table II: Average bounding box sizes for multi-vpin nets for benchmark `superblue1` at various split levels

When running our proximity attack, we did so from the attacker's point of view, limiting ourselves to BEOL information and the general trend knowledge of average bounding box size that can be gained from analyzing benchmarks (or similar circuit designs). As seen from the point of view of the attacker, 4-vpin nets would generally appear as two separate nets that each have 2 vpins at the split level, connected together down to layer one or connected with a u-shaped connection. For these nets, the attacker could use an average bounding box estimated from similar circuit designs or benchmarks. For higher number multi-vpin nets, an attacker could scale up the bounding box to account for benchmark analysis that shows that higher number multi-vpin nets have a larger average bounding box.

Previously, we identified the crouting proximity method as the most effective way to define a search area for potential matches in the 2-vpin net case. Recall this method took routing congestion into account to define the search area around each

vpin. Based on these results, we will also expand the search area for multi-vpin nets using the crouting proximity method. For the multi-vpin case, the crouting method will be applied such that the ratio of the number of vpins to the search area is equal across all the multi-vpin nets in the split level. If a multi-vpin net is in an area with high routing congestion, then the search area will be expanded (uniformly from all directions) until the *vpin density* in the new search area reaches a target value or the search area grows to four times its starting value.

To determine if a correct match is made in the multi-vpin case, we analyzed how multi-vpin nets are connected above the split level. There are two general ways a multi-vpin net can be connected: each individual vpin has only one direct connection to its matching vpin or 3 or more vpins are connected together directly. Our analysis of the benchmark shows that in the vast majority of cases for split layer 4 and higher, vpins in a multi-vpin net have only a direct connection to one matching vpin. This happened 92.0% when split at level 4, 98.4% when split at level 6, and 99.9% at level 8. Because a direct connection between two vpins is by far the most common case for higher split levels, our algorithm only tries to match each vpin to exactly one other vpin. For split level 2, only 62.1% of the vpins in multi-vpin nets are directly connected to exactly one matching vpin. We note the effect of our algorithm only trying to match to exactly one other vpin on the multi-vpin results for split level 2 in Section 3.3.

Simulation results comparing different proximity techniques as well as 2-pin and multi-pin nets are reported in Section 3.3.

3.2 Artificial Blockage Insertion to Combat Proximity Attacks

3.2.1 Blockage Insertion Procedure

To protect a design against proximity attack, we propose to alter it by artificially adding routing blockages at the target split level during the global routing stage. First, the global router generates an initial solution. Next, artificial blockages are

added to the design after analyzing the global routing solution and considering the target split level. The global router is called again to reroute the design which now includes routing blockage at the desired split level[2]. This is done using the same global routing tool. This time the router has to avoid the generated blockages. Once a new routing solution is generated, the artificial blockages are removed.

By adding the routing blockages at the target split level, the global router will be constrained to avoid the blockage areas, thus it will be forced to route more nets in the layers above the split level. This is because in practice, the routing algorithms try to utilize the lower levels as much as possible before moving to the higher layers (except for special cases such as routing timing-critical nets, etc.). Therefore, assuming the lower layers are already congested, the artificial blockages will force the router to move the routes to the layers above the split level. This intuitively results in additional vias, thus more vpins at the split level compared to the without-blockage case. We show proximity attacks get significantly more challenging because of blockage insertion.

A major consideration when introducing routing blockages is ensuring that a previously-routable design remains routable after blockage insertion. For example if too many blockages are inserted, the global router may simply not be able to route many of the nets without overflow of routing resources. Therefore we first discuss an algorithm to insert routing blockages while ensuring routability of the design.

Algorithm 1 gives the pseudo-code of our blockage insertion technique. For a given split level ℓ, we first divide metal layer ℓ into equal-sized non-overlapping rectangular windows. In our experiments for the ISPD'11 benchmarks [42], the window size was set to 100x100 which approximately created 100 windows per design.

[2]We note modern global routing tools have the capability to handle routing blockages at any layer.

Algorithm 1 Blockage insertion

1: **procedure** BLOCKAGE-INSERTION(SPLIT LEVEL ℓ)
2: divide layer ℓ into non-overlapping windows
3: **for** each window w **do**
4: avail_cap=0
5: **for** each global routing edge e in w **do**
6: avail_cap += $\max(0, cap(e) - util(e))$
7: s = 0
8: **while** (s+1)×(s+1) $\leq \frac{\texttt{avail\_cap}}{cap(e)}$ **do**
9: s ++
10: insert an s×s blockage at lower-left of w

Next for each considered window w, a square-shaped blockage is inserted in the lower-left corner of w. The area of the blockage is computed based on the existing routing congestion within w. Specifically, for each window w, we first compute the available, unused routing capacity inside w (in lines 5-7 of Algorithm 1). This is done by computing the available capacity of each edge in the global routing grid-graph and then adding the available per-edge capacity over all the edges inside w.

Line 6 of Algorithm 1 shows how the available capacity is computed for one edge. We subtract the routing utilization of the edge from its default capacity[3]. (Routing utilization of edge e is the number of global routes passing from *e* which is calculated from the initial global routing solution.) If this quantity is negative, the available capacity will be set to 0. Otherwise it will be equal to this quantity. This is shown in Algorithm 1 where the default capacity and utilization of edge *e* are denoted by *cap(e)* and *util(e)*, respectively.

Next, the blockage is found as a square with s×s grid edges, where s is set as the closest integer to the quantity avail_cap/*cap(e)*. This quantity estimates the

[3]By definition, all edges in the global routing grid-graph which are on the same layer have the same default capacity.

number of available grid-edges inside window w. The blockage is then inserted in the lower-left corner of w. The pseudo-code of the above procedure is listed in lines 8-10 of Algorithm 1.

Algorithm 1 ensures that the blockage inserted in each window is roughly the available routing capacity in that window. Therefore we expect that the global router should continue to generate a routable solution. However, because the blockage is inserted at the lower-left corner of each window (which includes some routed nets in the original solution), the situation will be different for the router in its new run. Therefore after inserting the artificial blockages, the router will be forced to route the nets in a different way to avoid the blockages. Since layer ℓ now has more blockage, and given that in practice the lower layers have a higher congestion, the router will be forced to route more nets to the higher layers, as we verify in our experiments in the next section.

Please note Algorithm 1 uses the existing, unused, routing resources in the design to insert the routing blockages. Therefore, it does not cause any area overhead to the design.

Moreover, it is not necessary for the blockage to be at the lower-left corner of *w* and it can be placed at random locations within *w*. But it should not make much difference because the number of windows is fairly large and each window is small in size relative to the overall routing grid dimensions. In fact, what impacts wirelength is ensuring that the window size *w* is a reasonable value. In practice, if *w* is too large, then the $s \times s$ blockage window within *w* will naturally be larger. A larger-sized rectangular blockage will be more disruptive to the same routing algorithm, i.e., it creates a routing solution that is more divergent, and potentially results in more wirelength overhead because of the need to detour around a larger blockage.

Algorithm 2 Timing aware blockage insertion

1: **procedure** TIMING-AWARE BLOCKAGE-INSERTION (SPLIT LEVEL ℓ, TIMING-CRITICAL NETS N_{crit})
2: **for** each net n in N_{crit} **do**
3: **for** each global routing edge e in $route(n)$ **do**
4: $cap(e) = cap(e) - 1$
5: $N = N \backslash N_{crit}$
6: Blockage-Insertion(Split Level ℓ)

3.2.2 Extension for Timing-Critical Nets

The discussed blockage insertion procedure was designed to preserve the usage of routing resources without increase in area. Here we discuss its extension to preserve a set of special nets. The algorithm guarantees that the routes of these nets remain the same as before blockage insertion. Therefore in the case when the nets belong to critical paths, blockage insertion guarantees that the delay on these paths remain intact.

The extended version of the algorithm, shown in Algorithm 2, requires that the set of timing-critical nets are specified beforehand and provided to it. (This can easily be done by doing static timing analysis on the version of the layout prior to blockage insertion.) Next, it translates the routes of the timing-critical nets into routing blockage. This ensures the routing resources corresponding to these nets remain reserved and can be used after blockage insertion to route these nets exactly as they were routed before.

More specifically, to implement a routing blockage, we first reduce the capacities of those edges in the global routing grid-graph which are used to route the timing-critical nets. This is shown in lines 2-4 of Algorithm 2 by visiting each timing-critical net and decrementing the capacity of each edge in its route. Next, timing critical nets are removed from the set of nets N. Algorithm 2 then uses this

updated grid-graph (with reduced capacities) and updated set of nets N to insert the blockage according to the procedure given in Algorithm 1.

After blockage insertion, the router is called to route the set of nets N (excluding the timing-critical nets). After the routing is finished, the routes of timing-critical nets are added to the routing solution of the remaining nets to create a full solution.

The above procedure ensures the required routing capacity for timing-critical nets are preserved and these nets remain routed just as they were before blockage insertion.

3.2.3 Comparison with Other Related Techniques

The main feature of our artificial blockage insertion technique is to force the router to route some nets with higher wirelength in order to detour the introduced blockage areas that were previously used. This in turn increases the vpin density at the non-blocked areas on the split layer, and thus makes the proximity attack more challenging from the perspective of each broken net by increasing the number of candidates in its neighborhood. Also, having fewer routing resources at the split layer will force the router to use the remaining, unused routing resources which are typically located on the higher layers because the lower layers have significantly higher routing congestion in modern designs [36].

At the same time our blockage insertion technique ensures the inserted amount of blockage is just the right amount to have a high chance to maintain routability of the design. This is because the inserted blockage is equal to the unused routing resources of the originally-routed design.

Increasing vpin density at the split level using our blockage insertion technique can be considered comparable with other related techniques to increase the vias at the split level. Therefore below we list other relevant techniques and then give a comparison with our artificial blockage insertion technique.

1) OBISA: In [47] the authors propose obfuscation of the layout by inserting Built-In-Self Authentication (OBISA) structures in the free spaces on the placement layout. The OBISAs are test structures that are made of the same standard cells already used to implement the design. Some nets are also carefully created to connect the OBISA structures to the design with guidelines to ensure timing constraints do not get violated and that testability is maintained. The above procedure is proposed as a means of obfuscation for split manufacturing that allows for a higher split level. While this procedure will certainly make the view of the untrusted foundry significantly more complicated, *nevertheless designers may not feel comfortable allowing the obfuscator to actually create connections to their designs.*

2) Adding Layer Assignment Constraints: Another method that can be applied to secure split manufacturing is for the user to add specific constraints to force desired nets to be routed on the higher metal layers. The router can then be evoked to obey these constraints. This method has the *disadvantage that the user will interfere with the layer assignment decisions which are done internally by the standard router.* The router uses sophisticated techniques for layer assignment and new automated algorithms are proposed to improve layer assignment, especially for modern technology nodes. The designer may not take all these considerations into account when specifying layer assignment constraints. Moreover, if too many layer assignment constraints are imposed by the designer to improve the obfuscation, it may impact the quality of the routed nets. This is a particularly important consideration because in modern technology nodes, higher layers are typically used to route timing-critical nets due to significantly lower delay on the top layers [21]. If the designer requires too many nets to be routed on the higher layers, the tool may not be able to satisfy timing constraints.

3) Use of Fewer Metal Layers: Another potential approach is to reduce the number of metal layers. As a result, the router will be forced to route the design with fewer routing resources, creating more congestion in the lower layers and certainly

making the view of the untrusted foundry more complicated. *However, in many instances of modern designs, the minimum number of metal layers are already used. Therefore eliminating even just the top-most metal layer can result in an unroutable design, which is unacceptable [22, 36].*

4) Branch Insertion: Finally, another option (that we propose in this work) is to add branches to some of the already-routed, non-critical nets in the design. For the routed nets that have existing timing slack, adding branches can be done as long as the length of the inserted branch does not exceed a maximum length beyond which the timing constraint will be violated. The branches are added by an obfuscating router, which keeps the existing routes intact and just utilizes the unused routing resources to create more routing congestion on as many nets as possible. Each branch starts from a point on a non-critical net and ends at the split layer. The end point of the branch is kept within a proximity of the projection of the start point on the split layer, and will act as a dummy via. (We will explain the details shortly.)

If a route is already connecting to the split layer, then adding a branch to it does not make a difference as far as its obfuscation. However it does make guessing the matching connection of the *other* broken routes more complicated. This is because by adding a branch to a route, other broken routes may now find the newly-added dummy via (corresponding to the branch) in their neighborhoods. In other words, the added branches appear as additional vpins from the prospective of other nets, which increases the set of candidates for those nets and makes the proximity attack more difficult. Moreover the obfuscator router may add a branch for those non-critical nets which were not broken before to connect them to the split layer and make them look like a broken net. Figure 3.3 illustrates these points.

Our branch insertion technique, similar to OBISA [47], has the disadvantage that it changes the original routed design by a likely-third party obfuscator. Designers

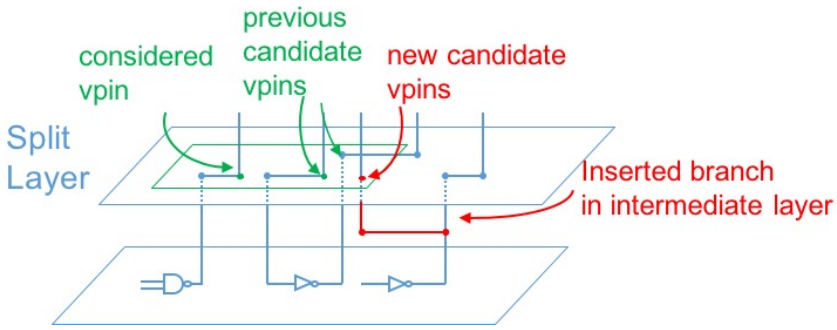

Figure 3.3: Branch insertion on one route can increase candidate vpin count for other broken routes, thus making the proximity attack more challenging.

may not feel comfortable using an obfuscator tool that essentially alters their designs, compared to the standard tools that the designers trusted to best consider various design issues.

To make the inserted branches unidentifiable, it is essential that the inserted branches have non-trivial topologies. The most trivial topology is when the start point of the branch connects to the split layer directly using a set of stacked-vias. To generate non-trivial topologies, we randomly pick the start point on the branch and set the end-point to be any point on a bounding box centered around the projection of the start point on the split layer.[4] We then use the shortest path algorithm to connect the start to any point on the bounding box. If the wirelength of the inserted branch exceeds the maximum wirelength for that start point, then we skip the route and move to the next non-critical route.

Once a branch is inserted for a route, the utilizations of the routing grid edges corresponding to the branch are incremented so that the available routing resources are correctly communicated when attempting to add a branch for the next net. Specifically, for the inserted branch, if the utilization of the edge becomes equal to the edge capacity, then the edge is removed from the grid so it cannot be used

[4]The bounding box size may be any of the placement, routing, crouting bounding boxes discussed in Section 3.1. We discuss more specific details based on our experimental setup in our simulation results when we compare branch insertion with our blockage insertion technique.

for any further branch insertion. The procedure stops when no new branches can be added for any non-critical routes. Algorithm 3 shows a pseudo-code for this technique.

Algorithm 3 Branch insertion

```
1: procedure BRANCH INSERTION (SPLIT LEVEL ℓ, TIMING-CRITICAL NETS N_crit)
2:     StopFlag = false
3:     for ∀ routing edge e in the routing grid do
4:         Remove e if util(e) = cap(e)
5:     while StopFlag == false do
6:         StopFlag = true
7:         for each net n ∉ N_crit do
8:             if branch not inserted for route(n) then
9:                 Randomly pick start point s on the route
10:                Compute W_max from start point
11:                Create bounding box B on the split layer
12:                Connect s to B with shortest path
13:            if wirelength(branch) ≤ W_max then
14:                Make branch permanent
15:                StopFlag = false
16:                for ∀ routing edge e ∈ branch(n) do
17:                    util(e) = util(e) + 1
18:                    if util(e) == cap(e) then
19:                        Remove e from the routing grid
```

Compared to the above techniques, our blockage insertion algorithm has the advantage that it does not modify the design by using an obfuscating tool. The same router is used to route the design and the router will make decisions such as

layer assignment and how critical nets should be routed. While the added artificial blockages challenge the router to create a different solution, these blockages are equal to the amount of unused resources of the originally-routed design. This makes it highly likely for the router to be able to generate an equally-good solution, but a different one which will in effect only *shift* some congestion from the lower layers to the higher layers. Moreover the proximity attack also gets more difficult after blockage insertion because the router is forced to detour around the artificial blockages (which were previously used). We will illustrate the above points in our simulation results using metrics to measure proximity attacks.

3.3 Experimental Results

In this section, we first compare and analyze our proposed proximity techniques. Next, we evaluate the effectiveness of our routing blockage insertion algorithm against proximity attacks. All our experiments were conducted using the ISPD 2011 routability-driven placement benchmarks [42]. The placements of these benchmarks were from the `UMich_SimPLR` tool which we obtained from the ISPD contest website. Next, an initial global routing solution was generated per benchmark using `NCTU-GR 2.0` or `CGRIP` [8, 22, 36]. In our experiments, we considered those benchmarks for which the router was able to generate a routable (i.e., 0 overflow) solution. These included 5 of the 8 benchmarks in the suite. There are no routable solutions to-date for the remaining 3 designs. Those were excluded from our experiments.

3.3.1 Comparison of Proximity Techniques

We implemented the proximity-based techniques discussed in Section 3.1. These included 5 techniques, i.e., proximity based on `placement`, `routing`, routing congestion (denoted by `crouting`), `union` of place & route, and `overlap` of place & route. Table III shows comparison between these 5 techniques for split levels 2, 4,

6, 8 for benchmark superblue1. This benchmark (and all others in suite) had 9 routing layers and due to lack of space we only list some of the split levels. In this experiment we considered the routing solution from the CGRIP tool[5].

Split level	Approach	#vpins	E[SA]	#HD matched	%HD matched	%match in list	E[LS]	FOM
2	placement	638236	30	34972	5.48	12.94	20.2	0.67
	routing	638236	24	4153	0.65	71.08	121.7	5.07
	overlap (P&R)	638236	54	25384	3.98	13.05	19.0	0.35
	union (P&R)	638236	54	3926	0.61	71.08	124.2	2.30
	crouting	638236	117	4153	0.65	82.08	515.7	4.42
	crouting (multi-vpin)	686463	323	1460	0.21	79.01	1697.6	5.26
4	placement	168602	289	2527	1.50	14.78	103.6	0.36
	routing	168602	196	1076	0.64	51.11	213.5	1.09
	overlap (P&R)	168602	485	1659	0.98	15.11	71.7	0.15
	union (P&R)	168602	485	982	0.58	51.46	251.5	0.52
	crouting	168602	755	1076	0.64	64.52	724.6	0.96
	crouting (multi-vpin)	91951	3230	964	1.05	70.72	1602.0	0.50
6	placement	47112	1156	36	0.08	16.53	212.0	0.18
	routing	47112	550	357	0.76	38.39	167.4	0.30
	overlap (P&R)	47112	1706	35	0.07	15.79	83.6	0.05
	union (P&R)	47112	1706	337	0.70	39.96	302.9	0.18
	crouting	47112	2082	357	0.76	53.61	562.1	0.27
	crouting (multi-vpin)	26549	6535	745	2.81	73.51	910.3	0.14
8	placement	7542	3721	5	0.07	11.54	142.4	0.04
	routing	7542	930	147	1.95	27.21	60.0	0.07
	overlap (P&R)	7542	4651	4	0.05	9.57	32.6	0.01
	union (P&R)	7542	4651	141	1.87	29.44	170.8	0.04
	crouting	7542	3537	147	1.95	42.97	182.8	0.05
	crouting (multi-vpin)	891	9734	208	23.34	75.08	60.6	0.01

Table III: Comparison of various proximity-based techniques for benchmarks superblue1 for various split levels

Furthermore, to compare different proximity techniques we considered both 2-vpin nets and multi-vpin nets. 2-vpin nets were those which had two cut-wires at the split level, multi-vpin nets had 3 or more cut-wires at the split level. For each vpin in a 2-vpin net, we defined a search area based on each proximity technique

[5]This is identical to the experiment reported in our ICCAD 2016 paper.

as discussed in Section 3.1. We also implemented a simple attack that for each considered vpin, it picked the vpin candidate from the search area which was in its closest proximity (in terms of Manhattan Distance). For multi-vpin nets, we defined a search area using a larger bounding box as discussed in Section 3.1.

Specifically, in Table III, we evaluate the following metrics per technique per split level.

- **#vpins:** This is the number of vpins reported at each split level.
- **E[SA]:** This is the average size of Search Area (SA) per technique. The area is measured in terms of number of global cells. The average is done over all SAs considered at a considered split level.
- **#(%)HD matched:** These two metrics measure the number (and percentage) of vpins for which we successfully found the matching cut-wire using the proximity attack of picking the closest vpin (based on Hamming Distance) in the search area.
- **E[LS]:** This metric reports the candidate List Size (LS), averaged over different search areas. In other words this metric measures the average number of candidate vpins that is identified in each SA. The lower value of this metric indicates an easier job for the attacker to apply various pruning strategies.
- **%match in list:** This metric reports the percentage of vpins for which the actual matching vpin was included in candidate list found for each SA. It measures the effectiveness of our proximity technique in narrowing down the promising candidates for each cut-wire to a smaller set.
- **FOM:** This is a Figure of Merit defined as E[LS/SA]. It is the ratio of candidate list size divided by the search area, when averaged over all the search areas at a split level. We discuss later that a higher value of this metric means

it is more challenging for the attacker because the density of candidates (over the same search area) will be higher.

Based on the above metrics, we make the following observations from Table III.

1) Success in finding a match for 2-vpin and multi-vpin nets: In terms of percentage of finding the match in candidate list, we observe that `crouting` has the highest percentage, followed by `routing` and `union` techniques (which had similar values), followed by `overlap` and finally the `placement` techniques. For example at split level 2, the percentages of finding the correct match among the candidate list were 82.08%, 71.08%, 71.08%, 13.05% and 12.94% for `crouting`, `union`, `routing`, `overlap` and `placement` techniques for 2-vpin nets, respectively. We can see this is in part because `crouting` has the highest average search area (E[SA]) which likely translates into a larger candidate list size, thus more chance to include the matching vpin. For a similar reason the `overlap` technique had the lowest value.

The lower percentages of other techniques compared to `crouting` indicates that there are many cases when the matching vpin is not in close proximity and considering routing congestion is an effective way to include the actual matching vpin in the list of candidates.

In terms of dependency on the split level for 2-vpin nets, interestingly the percentages usually drop as we move to the higher layers. For example at split level 8, the percentage drops to 42.97% in the `crouting` technique which is almost half of the one in split level 2. This just means that the effectiveness of these proximity techniques are lower in the higher layers in including the matching vpin in the candidate list. This is because at the higher layers, the wire sizes are typically wider so the average list size (E[LS]) becomes smaller, thus the chance of including the matching vpin in the list decreases. For example for `crouting` the E[LS] drops from 515.7 in split level 2 to 182.8 at split level 8. This dependency did not hold true for multi-vpin nets, where we saw a fairly consistent value for percentage

match in list between 70% and 80%. For 2-vpin nets, the values of `routing` and `union` were very close or the same within a split level and better than `placement`. This indicates that the `routing` technique is more effective than `placement`.

Despite the relatively high percentages of `crouting` in including the matching vpin in the candidate list, we note the actual match rate (and #matches) are quite low across all techniques and split levels for both 2-vpin and multi-vpin nets. The one exception to this was the multi-vpin net case for split level 8 where we saw the percentage matched value spike to 23.3%. This outlier value may have resulted from the relatively small number of multi-vpin net vpins on that level, 891, making a proximity attack more likely to select the correct match. Overall, these results indicate that proximity alone is in no way sufficient to solve the reverse engineering problem. However the significantly smaller list size (compared to total #vpins) is still quite useful in narrowing the candidates to allow aggressively applying various pruning rules, which falls beyond the scope of this work.

Finally, we note that for multi-vpin nets at split level 2, 37.9% of vpins are connected to more than one other vpin. For this type of connection, we assume a correct match when the proximity algorithm finds a match with one of the other matching vpins. This is the only case where there are a significant number of connections of this type as opposed to all other cases where the vast majority of vpins have exactly one match. This assumption skews the # and % matched results higher for the split level 2 multi-vpin case, but has a negligible effect on all other multi-vpin results.

2) Comparison of FOM (difficulty of the attack): We concluded `crouting` was the most effective technique in including the matching vpin in the candidate list due to its larger list size. Here we are interested to see if this larger list size means a more difficult attack (higher value of FOM). Note FOM in effect measures the density of candidates (in the search area). So a lower value means there are fewer candidates in the same search area which can be interpreted as less challenging from the attacker's perspective. We observe `crouting` and `routing` techniques both had

higher FOMs compared to `union`, `placement`, and `overlap`, respectively. This is an interesting observation because `crouting` had a larger search area as well as a larger list of candidates, and overall it also had higher FOM.

In terms of dependency on split level, we observe that all FOMs significantly drop when moving to the higher layers. While E[LS] is smallest in the highest split level, FOM is smallest as well, so the attack becomes easier.

3.3.2 Comparison with More Benchmarks

Recall, each of the search areas defined by each our proximity techniques was essentially a function of either a base placement bounding box or a base routing bounding box value per split layer. Before analyzing our proximity techniques for other benchmarks, we first explain how these based bounding box values are computed. We first computed the average bounding boxes (for both placement and routing) of 2-vpin nets. Table IV reports the average bounding box sizes for two-vpin nets at various splits levels for these benchmarks using the `NCTU-GR` routing tool[6].

As can be seen the bounding boxes increase as the split layer increases. We then chose the overall average across all the benchmarks reported in Table IV per split level as reported in the final row for levels 2, 4, 6, and 8, as the base values when running our proximity attacks. *Therefore we used the same single placement bounding box and single routing bounding box per split level as base values across all the benchmarks. We note the actual search areas vary across the techniques (and possibly per broken net) and they are defined as a function of these base values.* We consider these base values as representative bounding boxes of typical designs.

Based on these base values, next we report the results of our proximity techniques in Table V for other benchmarks. Due to lack of space we only report the

[6]For this experiment we switched to the NCTU-GR routing tool because CGRIP did not generate routable solution on the remaining benchmarks.

Split Level	2		4		6		8	
Routing bounding box								
	E[BB]	SDV[BB]	E[BB]	SDV[BB]	E[BB]	SDV[BB]	E[BB]	SDV[BB]
superblue1	19.28	67.45	77.50	136.34	146.19	162.85	207.34	147.55
superblue5	21.46	65.89	70.19	107.67	108.50	127.25	134.76	124.86
superblue10	31.22	122.70	96.04	188.37	116.54	177.22	125.85	139.60
superblue12	9.67	25.78	32.78	51.66	37.74	63.02	36.17	59.85
superblue18	17.75	56.05	43.81	73.02	58.12	84.24	60.61	85.45
overall average	19.88		64.06		93.42		112.95	
Placement bounding box								
	E[BB]	SDV[BB]	E[BB]	SDV[BB]	E[BB]	SDV[BB]	E[BB]	SDV[BB]
superblue1	21.26	68.01	85.57	137.00	185.96	175.90	330.38	221.40
superblue5	23.10	66.42	78.46	110.58	148.15	138.57	243.88	181.82
superblue10	33.58	123.41	108.17	196.56	162.06	214.39	281.05	289.98
superblue12	11.33	26.17	41.05	55.56	53.76	75.03	71.07	96.95
superblue18	19.81	55.91	52.10	74.25	81.57	97.24	116.35	131.64
overall average	21.82		73.07		126.30		208.55	

Table IV: Average bounding boxes for various benchmarks, including standard deviation

results for the `union` and `crouting` cases per benchmark, which were the top performing proximity techniques. These results follow a similar trend as observed in Table III.

3.3.3 Impact of Blockage Insertion

In this experiment we implemented the blockage insertion algorithm presented in Section 3.2 and applied it to the global routing solution of each benchmark assuming the split level is 4. We first consider the standard version given by algorithm 1 and later present the results for the variation which considers timing-critical nets. The results are shown in Table VI and generated based on the `NCTU-GR` routing tool.

Benchmark	Split Level	Approach	#vpins	E[SA]	E[LS]	%match in list	FOM
superblue1	2	union	644888	60.0	126.6	71.20	2.11
	4		145666	632.0	338.6	45.53	0.54
	8		7332	3568.0	124.3	11.10	0.03
	2	crouting	644888	98.7	436.3	81.45	4.42
	4		145666	1112.7	885.6	58.19	0.80
	8		7332	3113.1	115.1	15.53	0.04
superblue5	2	union	695640	60.0	142.0	64.32	2.37
	4		173216	632.0	284.7	35.10	0.45
	8		10940	3568.0	145.8	26.45	0.04
	2	crouting	695640	96.7	477.0	76.85	4.93
	4		173216	1106.8	745.8	53.70	0.67
	8		10940	3086.7	149.4	35.63	0.05
superblue10	2	union	937176	60.0	167.4	59.87	2.79
	4		192562	632.0	438.9	46.40	0.69
	8		14772	3568.0	182.5	33.89	0.05
	2	crouting	937176	96.9	595.5	78.34	6.15
	4		192562	1060.9	939.4	54.68	0.89
	8		14772	2934.9	185.4	42.45	0.06
superblue12	2	union	1175670	60.0	224.8	68.25	3.75
	4		165186	632.0	745.7	61.09	1.18
	8		15966	3568.0	773.7	72.42	0.22
	2	crouting	1175670	99.6	876.8	85.46	8.80
	4		165186	1106.2	2078.8	75.67	1.88
	8		15966	2523.8	870.4	73.13	0.34
superblue18	2	union	369632	60.0	151.5	53.28	2.53
	4		86760	632.0	428.0	45.69	0.68
	8		6968	3568.0	250.0	55.37	0.07
	2	crouting	369632	98.8	512.3	73.03	5.19
	4		86760	1100.1	1076.9	70.13	0.98
	8		6968	3173.7	280.7	66.88	0.09

Table V: Comparison of proximity-based union and crouting techniques for various benchmarks

The last 4 columns report the average list size (E[LS]) and FOM for before and after blockage insertion cases. As can be seen both E[LS] and FOM increase in the after-case. This indicates there are a higher number of vpin candidates, and the difficulty of attack in terms of FOM also increases by inserting blockages. Also in

	Comparison of vias at each layer								Comparison of proximity metrics			
	Before	After (difference in vias relative to Before case)							Before		After	
	Total Vias	V23	V34	V45	V56	V67	V78	V89	E[LS]	FOM	E[LS]	FOM
superblue1	4597616	+48672	-17677	+40051	+70355	+20743	+16950	+2940	481.4	1.4	601.1	2.2
superblue5	4650756	+66498	-17050	+34828	+62704	+31504	+24059	+8668	694.8	1.7	838.8	1.9
superblue10	6304110	+54895	+15192	+42210	+50999	+38102	+25112	+16125	943.4	2.2	1099.9	2.4
superblue12	8913075	+132068	+96096	+151018	+175614	+128236	+91327	+61938	1876.7	3.7	2383.5	4.2
superblue18	3582687	+57475	+3367	+45417	+72897	+37020	+31311	+15635	966.2	2.3	1347.3	2.7
average									992.5	2.2	1254.1	2.7

Table VI: Comparison of various metrics for before and after block insertion cases for split level 4

this experiment we used the routing proximity technique with the average routing bounding box of 65 (rounded up to the next whole number from 64.06), as shown in Table IV. Moreover since blockage insertion is concerned with the overall number of vpins on the split level and not just with 2 vpin nets, we used the multi-vpin approach explained in Section 3.1.3 to identify candidates for our evaluations in this experiment.

Finally, in columns 2-9 we report the total number of vias in the design (for the before-case) and *additional* inserted vias in the after-case. For the after-case, we report the breakdown of via count for all via types. So for split level 4, the V45 vias are actually the number of additional vpins added on the split level. As can be seen, insertion of blockage always results in increase in number of used vias in the split level and the layers above it. This indicates more nets are routed in the higher layers. For the layers below the split level, in some benchmarks (superblue1, superblue5) the number of vias decreases on layer 3 but in the rest of the benchmarks the number of vias also increases in the lower layers. Finally, we would like to emphasize that all benchmarks remain routable after blockage insertion and our algorithm preserved the routability feature of each benchmark.

Figure 3.4 shows the map of the difference in number of vpins on split level M4 for benchmark superblue1 for the before and after blockage insertion cases. The *additional* number of vpins in this layer is 40,051 in the after-case. These additional vpins are actually inserted in various layout areas. The *difference* varied from -10 to 10 at each location.

In our next experiment we consider the variation of the algorithm to preserve a set of special nets which can be considered timing-critical. Here for each benchmark we selected 10% of the nets at random as timing-critical and applied the extended version of the algorithm as discussed in Section 3.2 using the NCTU-GR routing tool. Similar to the previous experiments, we assumed split level is M4 for blockage insertion.

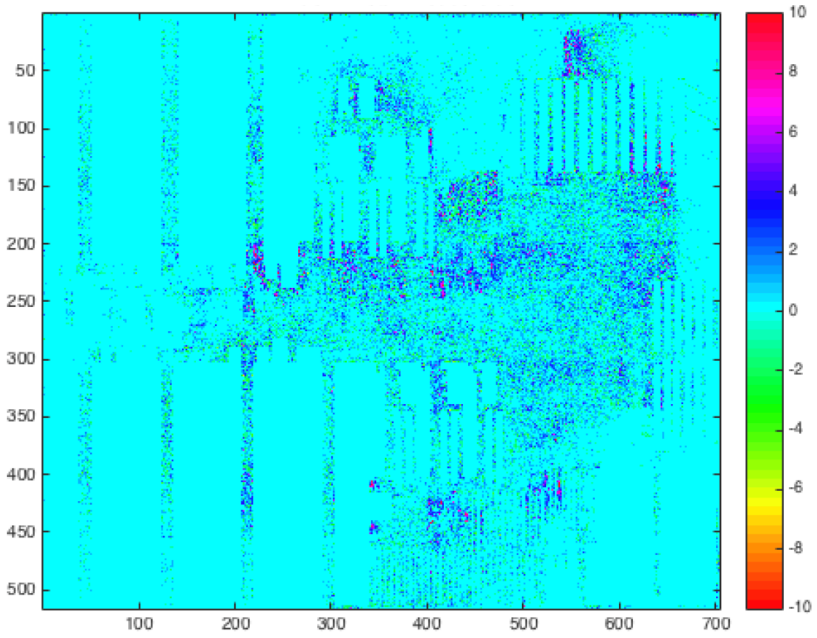

Figure 3.4: "Difference Map" showing the increase or decrease in number of vias (vpins) after blockage insertion at various regions of the split layer.

The results are reported in Table VII. Columns 2-8 report the percentage of vias which belonged to the timing-critical nets compared to the total number of vias for each via type. For example at M4 (split layer), for benchmark `superblue1`, the number of V45 which belonged to timing-critical nets were 14.73% of total number of V45s. Columns 9-15 report the percentage increase in the total number of vias after blockage insertion for each benchmark. These totals are for all nets, including timing-critical nets. For example in `superblue1`, the number of V45s increased by 19.18%. Note the number of V45s which belonged to timing-critical nets remained intact. But for the rest of the nets there was a significant increase at the split level in number of vias. This significant increase in number of vias can also be observed as we move to higher layers, indicating that the algorithm tends to route more nets on the higher layer after blockage insertion.

The last 2 columns of Table VII report the average list size (E[LS]) and FOM for the after blockage insertion case. As can be seen both E[LS] and FOM increase

	% Vias of timing-critical nets (Before Blockage Insertion)							% Vias (After Blockage Insertion)							Proximity metrics	
	V23	**V34**	**V45**	**V56**	**V67**	**V78**	**V89**	**V23**	**V34**	**V45**	**V56**	**V67**	**V78**	**V89**	**E[LS]**	**FOM**
superblue1	8.32	11.89	14.73	20.29	13.16	21.13	9.24	3.79	-6.97	19.18	80.10	23.28	65.07	6.19	633.1	1.6
superblue5	8.12	8.80	9.01	11.43	7.66	8.85	5.24	5.07	-5.71	9.43	29.84	12.74	24.01	6.45	861.7	1.0
superblue10	9.71	4.79	3.01	2.92	1.49	1.57	0.92	10.61	7.14	26.05	54.18	64.85	84.09	113.69	1431.3	2.8
superblue12	8.83	7.79	7.82	9.42	8.80	10.50	10.34	3.41	0.07	9.72	32.40	16.99	35.59	21.73	2232.5	4.2
superblue18	6.69	5.21	5.11	7.25	4.10	6.46	2.02	5.90	-2.38	14.32	47.41	24.73	58.66	37.57	1298.2	2.6
average															1291.4	2.6

Table VII: Comparison when the routes of 10% of the nets are preserved due to being timing-critical

in the after-case as compared to the before-case reported in Table VI. This indicates there are a higher number of vpin candidates, and the numbers are comparable to the after-case reported in Table VI where blockages were inserted and timing-critical nets were not considered. This experiment also used the multi-vpin approach to identify candidates explained in Section 3.1.3 and the `routing` proximity technique with the average routing bounding box of 65, as was done for the results in Table VI.

For some benchmarks like `superblue10`, it is interesting to note that the percentage increase in V45 is still very high (26.05%) despite the small percentage of V45s which belong to timing-critical nets (only 3%).

3.3.4 Comparison with Branch Insertion Technique

In our final experiment, we make a comparison between the branch insertion and blockage insertion techniques. Here branch insertion serves as an alternative which adds dummy vias so it can provide a point of reference for comparison.

In this experiment we used the `NCTU-GR` routing solutions for various benchmarks. For the blockage insertion technique we used the timing-aware variation with the setup identical to our previous timing-aware blockage insertion experiment. For branch insertion, we used the same nets that were considered timing-critical as in blockage insertion technique and did not insert a branch on those nets.

We used Algorithm 2 for branch insertion. However we did not use a wirelength constraint when adding a branch for a non-critical net so W_{max} in Algorithm 2 was set to a very large number. This results in adding a higher number of branches and provides an optimistic estimate of branch insertion technique. We show this optimistic estimate provides comparable values of the evaluation metrics as in the blockage insertion approach. We also compare both techniques with the original one (i.e., the original `NCTU-GR` routing solution).

Similar to the previous experiments, we experimented when the split layer is metal 4. For this split layer we used the same value of bounding box (overall average of 65) as reported in Table IV to compute all the evaluation metrics across the benchmarks. We also used the multi-vpin approach explained in Section 3.1.3 to identify candidate lists in this experiment.

As can be seen from Table VIII, branch insertion results in better (higher) E[LS] and FOM compared to the original routing solutions so it works as an obfuscator. This is because adding branches in unbroken routes will make them appear as broken. Moreover, adding branches to broken routes will complicate the view of other broken routes and can result in increasing the number of vpins in the proximity of *other* broken routes.

	Original		**Timing-Aware Blockage**		**Branch Insertion**	
	E[LS]	**FOM**	**E[LS]**	**FOM**	**E[LS]**	**FOM**
superblue1	481.4	1.4	633.1	1.6	611.8	1.9
superblue5	694.8	1.7	861.6	2.0	790.8	3.0
superblue10	943.7	2.2	1431.3	2.8	958.5	3.5
superblue12	1876.7	3.7	2232.5	4.2	2021.9	4.7
superblue18	966.2	2.3	1298.2	2.6	1074.6	2.8
average	992.5	2.3	1291.4	2.6	1091.5	3.1

Table VIII: Comparison with branch insertion obfuscator

When comparing blockage insertion with branch insertion, we see blockage insertion has somewhat better E[LS] while branch insertion has somewhat better FOM metric. Therefore we conclude both blockage insertion and branch insertion are comparable and both help increase the proximity attack difficulty compared to the original routing solution. The advantage of blockage insertion is that the same router is used to route the design after artificial blockage insertion. However in branch insertion, an obfuscator routing tool needs to add these additional branches.

We note, comparison of number of vias between blockage insertion and branch insertion techniques is not fair. Branch insertion generally has many more vias than blockage insertion. However, we observed that many of the vias on the split layer connect to one more via through the inserted branch and can be easily identified. This is not the case for the blockage insertion technique.

3.4 Conclusions

Building on existing research, we have developed new techniques to define proximity based on both placement and routing information. For each technique we introduced intuitive ways to define the search neighborhood around each vpin. We also proposed a technique for blockage insertion at the split level and showed its impact in decreasing the effectiveness of proximity attacks.

In our experiments we concluded that proximity alone is no way sufficient to reverse engineer the FEOL and guess the connections captured by BEOL. However, we showed our proximity techniques (especially `crouting`) can be quite effective in including the actual matching vpin, on-average by 62.74% for different split levels for 2-vpin nets across several benchmarks, and narrowing down the list of candidates to significantly small size.

Having established that `crouting` was the most effective of our novel proximity-based techniques, we have shown that it is important to define the search area as a function of routing congestion which results in different search areas per vpin, per split level.

Chapter 4

Further Improvement Using Machine Learning

In this chapter we use machine learning to improve the proximity attack from Chapter 3 and [24, 25]. We use the identical setup but include more layout features when analyzing the attack and ensure separation of training and testing data sets. Similar to 3, the emphasis is to find a small list of candidates for each vpin (broken net) which includes the actual match in the list with high accuracy.

Since for lower split layers our basic machine learning algorithm's runtime becomes prohibitively large, we propose novel approaches to make it scalable. Without sacrificing much in the effectiveness of our attack, we achieve significant runtime improvement. We then rank the features used by our machine learning algorithm by importance, and study how this ranking changes across the layers of the design. We show routing features are most important, but that as we move to lower layers other features become important as well.

As in Chapter 3, our experiments in this chapter are conducted based on data collected from design layouts released by industry [42], featuring 9 metal layers with significant variation in wire size and significant variation in routing congestion across the layers.

4.1 Machine Learning Framework

Figure 4.1 shows a high-level overview of our modeling framework. First, a challenge instance is created from a placed and routed layout by cutting it at the split

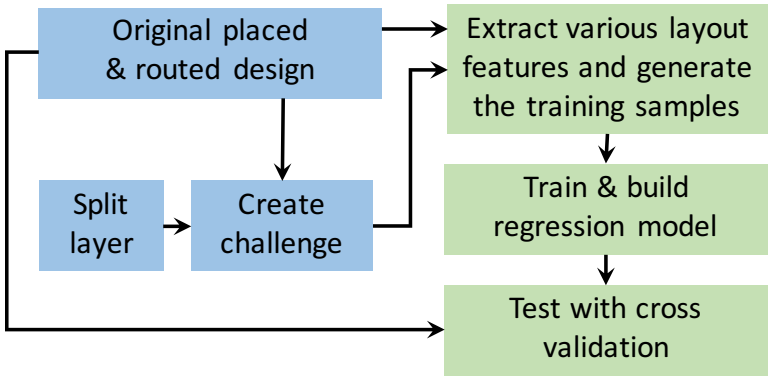

Figure 4.1: Flow chart of our modeling process

layer and only looking at the FEOL view. Next, a set of layout features are collected for each vpin, which include features from placement, routing, cell areas, and cell pin types. Using these features, next, machine learning samples are generated which are fed into the training process. Each sample includes information for a *pair* of vpins which may or may *not* be matching each other. (This step will be explained in detail later.) Sample generation is done on a subset of the designs which are designated for training. Machine learning then builds a classifier using the training samples. Cross validation is used for evaluation which ensures validation of the model is done on the rest of the data which was not used for training.

We first explain how the layout features are collected for individual vpins, then explain how samples are generated for training, and discuss details of the modeling process. Finally, we discuss adjustments to the machine learning framework to make it scalable when applying to lower split layers.

4.1.1 Feature Extraction for Individual vpins

For each vpin *v* (in the designs used for training), we extract the following layout features as shown in Figure 4.2. Let (vx, vy) denote the vpin's coordinates on the split layer. We compute wirelength *W* for the route fragment that connects v to one or more pins of standard cells on the underneath placement layer. We also calculate

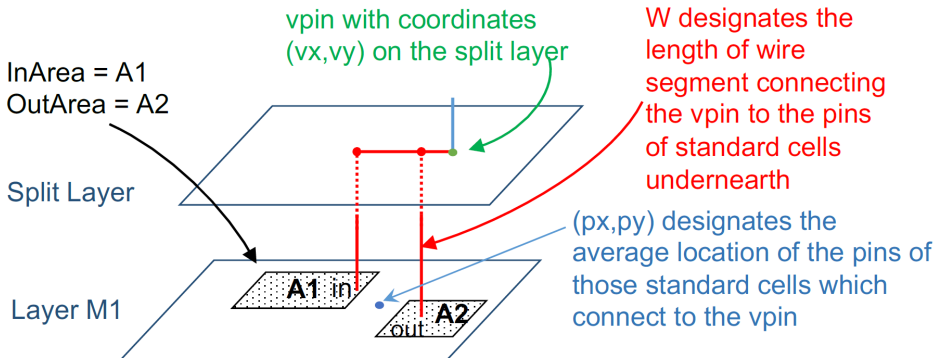

Figure 4.2: Layout feature extraction for each vpin

the location where the vpin connects on the placement layer, which we denote by (px, py). If the connection is to multiple pins on the placement layer, the location is computed by averaging the coordinates of pins of the standard cells that connect to v.

Among the standard cells that connect to v, we sum the areas of those which connect to v through an input pin. We denote this by `InArea`. Similarly, we compute an `OutArea` designating the sum of areas of the standard cells which connect to v through an output pin. Figure 4.2 shows an example of how the above quantities are calculated. We note these features together capture characteristics from placement, routing, cell areas, and pin types.

The reason `InArea` and `OutArea` are defined is to enable accounting for driver strength during machine learning. Driver strength is highly correlated with the cell area. Each cell can drive a maximum output load. These will be used during machine learning when a pair of vpins are considered as potential match. Similarly, the reason wirelength W is recorded is to enable identifying cases when a pair of considered vpins result in a realistic combined wirelength.

4.1.2 Sample Generation and More Details About Training and Testing Stages

After the layout features for individual vpins are extracted, we prepare the samples to feed into the subsequent training step. Specifically, for each vpin, we first create a pair by recording the index of its (correctly) matching vpin. This serves as a 'positive sample'. We then create equal number of samples of non-matching vpin pairs which we denote as 'negative samples'. This is done by randomly picking vpins which are not a match, from the training data set. (We explain later how we divide our benchmarks into testing and training data sets). This ensures equal number of matching versus non-matching samples in training which is shown to be essential for effective modeling when dealing with unbalanced data sets [19][1].

Next, for each pair (matching or non-matching), we record various corresponding layout features. This uses the individual vpin features which were explained before. Specifically, for a vpin pair (v_1, v_2) we record the following to generate a sample.

- `DiffVpinX` = $|vx_1 - vx_2|$: This feature records the difference in the x-coordinates of the two vpins. It captures the observation that a candidate vpin in close proximity is more likely to be a match [24, 25].
- `DiffVpinY` = $|vy_1 - vy_2|$: Same as the previous feature except calculated using the y-coordinates.
- `DiffPinX` = $|px_1 - px_2|$: This feature records the difference in the x-coordinates of the pins on the placement layer which connect to the two vpins.
- `DiffPinY` = $|py_1 - py_2|$: Same as the previous feature except calculated using the y-coordinates.

[1] Those vpin pairs which connect output pins of different cells are illegal and we exclude from both positive and negative samples.

- `HammingVpin` = $|vy_1 - vy_2| + |vx_1 - vx_2|$: This feature records the Hamming distance between two vpins. We note, including the difference in individual x and y coordinates, besides the Hamming distance as separate features allows machine learning more flexibility to consider proximity between the vpins as a factor in matching the correct vpin.

- `HammingPin` = $|py_1 - py_2| + |px_1 - px_2|$: Same as the previous feature except it is calculated based on the pin locations on the placement layer.

- `TotalWirelength` = W_1 + W_2: This is sum of the wirelengths associated with the two vpins. Since the wirelength of each net impacts timing, it is important to ensure the combined wirelength of the considered vpin pair is not prohibitively large (which will be decided by machine learning algorithm based on typical characteristics seen in the training data).

- `TotalArea` = $\texttt{InArea}_1$+$\texttt{InArea}_2$+$\texttt{OutArea}_1$+$\texttt{OutArea}_2$: This records sum of cell areas connecting to the two vpins.

- `DiffArea` = (`OutArea`$_1$+`OutArea`$_2$)-(`InArea`$_1$+`InArea`$_2$): This feature first records sum of cell areas which connect to the pair via an output pin, and via an input pin as two separate quantities shown in the parenthesis. It then subtracts these two quantities to record the area difference of the driver cells from its loads. Note, one of `OutArea`$_1$ or `OutArea`$_2$ must be zero, and otherwise the pair is illegal and is disregarded.

The `TotalArea` and `DiffArea` features together help determine if a driver cell has sufficient strength for the load introduced by the vpin pair. Figure 4.3 (a) and (b) illustrate some of the above items.

Overall, each sample represents a vpin pair and includes the above 9 layout features. Finally, a binary value is the last field included in a sample which indicates if the pair is a match (i.e., if it is a positive or negative sample). Using the generated

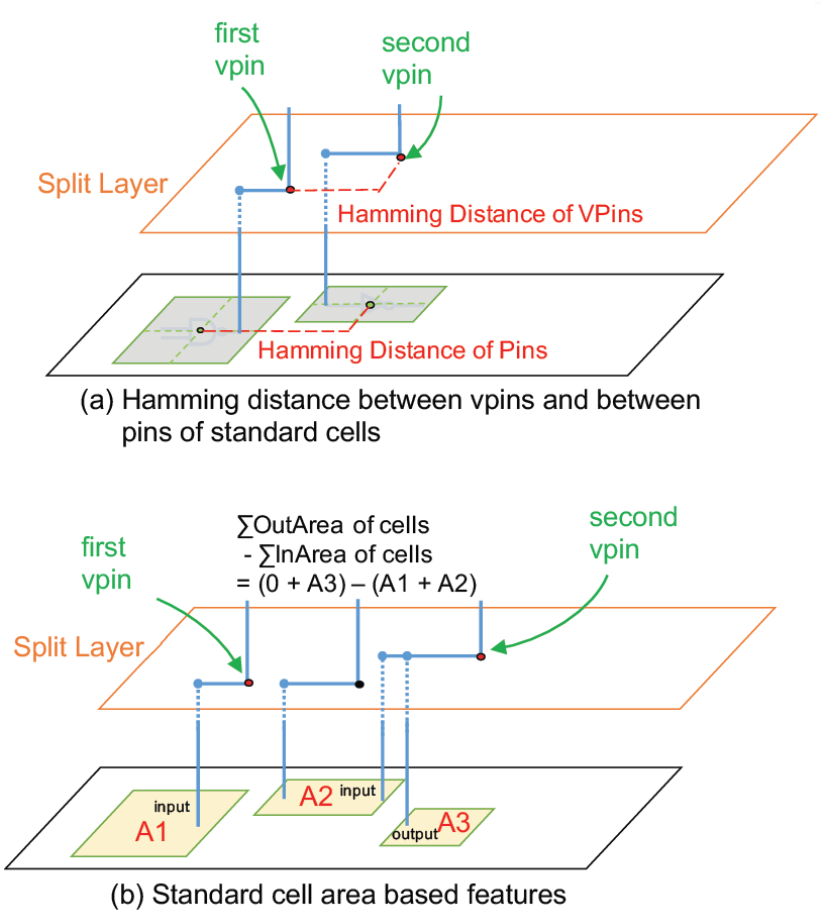

Figure 4.3: Illustration of some features for pairs of vpins

samples, next **training** is performed. For training, we experimented with 8 different classifier algorithms and found `RandomForest` to perform the best so we used it as the default classifier in our experiments. Due to lack of space we don't report the details of this comparison.

Once, the training is done, in order to do the **testing**, first the set of all possible unique vpin pairs are considered for evaluation. Specifically, for each vpin, all other vpins in that split layer are considered for evaluation; if n vpins exist on the split layer, up to $\binom{n}{2}$ pairs are evaluated[2]. The testing stage then solves the inference problem by generating a yes/no answer for each pair to predict if the two vpins are

[2]Note, pairs connecting two output pins are disregarded.

a match. Finally, in our experiments we use the 'leave-one-out' **cross validation** which is a standard procedure to ensure separation of testing and training data sets. Specifically, assuming we have k designs, to test each design, we use the $k-1$ remaining designs for training the model. This means a separate model is built per design (for a considered split layer) and different number of samples are generated per design.

Once the testing stage is finished, the matches (i.e., yes answers) for each vpin are bundled together into one set and recorded as its identified 'List of Candidates' which we denote by **LoC**. We measure the quality of our modeling with two metrics. First, we measure the size of the LoC. Clearly smaller-sized list is better. Second, we care about classification accuracy meaning that the LoC should include the actual match for the vpin with a high likelihood.

4.1.3 Improvements for Scalability

We faced **three issues** when applying our machine learning framework (denoted by ML) to the lower split layers. So we propose improvements to extend our framework to the lower layers. We denote this improved variation by ML-Imp.

First, we observed significant increase in the runtime of ML when applying it to the lower metal layers as the split level. This increase was both in the training and testing times (e.g., from on-average 8 minutes to 21.5 hours when switching from layer 8 to layer 6). This was because there were a significantly larger number of samples generated for training, and more pairs to evaluate during testing. For training, the positive samples increased because there were a much higher number of vpins located on the lower layers. (The number of negative samples also increased because they were equal to the positive ones.) The testing runtime also took longer because the number of possible vpin pairs to evaluate was significantly higher. The runtime of ML for layer 4 was prohibitively large requiring implementing improvements for scalability.

Second, when moving to lower layers, the classification accuracy of ML dropped (for example from on-average 99.97% to 90.90% when moving from layer 8 to 6). We observed this was due to having 'useless' negative samples during training. Recall the negative samples were selected by randomly picking two vpins which were not a match. However, often times we observed the vpin pair used as the negative sample were so far apart from each other on the split layer that their extracted layout features did not provide useful information for learning. In other words just looking at the `HammingVpin` was sufficient to predict the pair was not a match. Also the remaining layout features may have acted as unwanted noise and degraded the learning.

Finally, we observed that moving to the lower layers resulted in the size of the generated LoC to grew at a much higher rate. Recall the size of LoC designates the number of candidates identified by ML as match for each vpin. For a more effective attack we aim to minimize the size of LoC.

To address the above three issues, our main observation is to eliminate the useless negative samples during training. (See the second issue for explanation of useless samples.) Also, we aim to eliminate useless vpin pairs from testing; these can sometimes degrade the proximity attack by generating false positives and increase the size of LoC. Specifically, instead of considering all vpin pairs on the split layer for testing, and randomly picking the negative samples for training, we aim to consider a 'useful' subset for each vpin which are at a closer proximity to it. The LoC is then identified from this subset. We explain how such a proximity is modeled in ML-Imp shortly. The idea is that once a neighborhood is found, the samples for training (both positive *and* negative) and testing of ML are only taken from the neighborhood which is then used to generate LoC.

In general, we aim for the neighborhood size to be significantly smaller than the overall area of the split layer to reduce the number of tested vpin pairs as much as possible, thus improve runtime, and focus on useful samples so accuracy is not

impacted much. We note prior work [24, 25] also focused on identifying a small neighborhood around each vpin using linear regression, and declaring *all* vpins in the neighborhood in the LoC. Compared to that, our neighborhood size will be much larger and while ML-Imp considers all the vpins in the neighborhood, it ultimately selects a subset of the vpins to be included in the LoC.

Now we explain how the neighborhood is identified. As mentioned before we use leave-one-out cross validation for testing and training. To determine the neighborhood for one design, we first study the distribution of the `HammingVpin` feature in the remaining $k-1$ designs for the actual vpin pairs (that are true match). Specifically, for each vpin in the training data set, we already know its match and can measure the Hamming distance between the two. This is done for all vpins in the $k-1$ designs. Next, the neighborhood size is picked such that 90% of all these vpin pairs are included in it. The ML-Imp variation uses this neighborhood for training by only generating samples for the vpins included in it.

Figure 4.4 shows the Cumulative Distribution Function (CDF) for different designs for layer 6 as the split level. The curve for each design represents the aggregate data of the remaining $k-1$ designs. We emphasize that this process ensures the testing and training data are separated. Moreover, this step is done significantly faster than the testing and training because the number of actual vpin pairs (that are true match) are significantly smaller than all considered candidates. Finally, we note the 90% cutting point can be changed in order to trade off the runtime of ML-Imp with its classification accuracy. Defining the neighborhood based on a smaller percentage, say 80% can accelerate ML-Imp however it excludes (from the training process) the knowledge of those 20% of connected vpins which are further apart. So classification accuracy may slightly degrade during the testing stage.

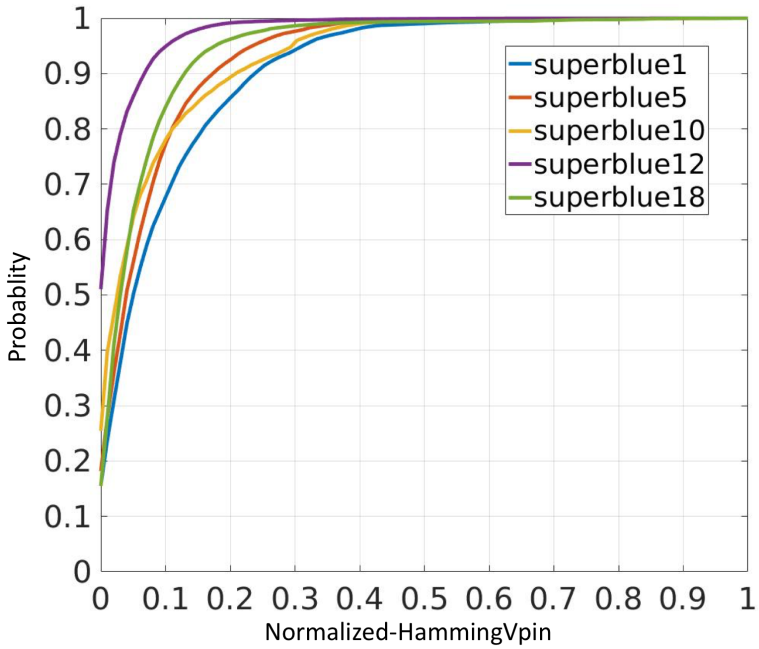

Figure 4.4: Cumulative distribution function of the (normalized) `HammingVpin` feature is shown for the truly-matching vpins in the training data set for each design.

4.2 Experimental Results

We experimented with the same setup as in Chapter 3 and [25][3]. We implemented the ML framework described in Figure 4.1 which first generated a challenge case for a given split layer, extracted layout features, prepared the samples, applied training and then testing using the 'leave-one-out' cross validation to ensure separation between the testing and training data sets. (Please refer to the end of Section 4.1.2 for detailed explanation of the leave-one-out cross validation.)

We used the Weka tool [9] for training and specifically used the `RandomForest` classifier as the default algorithm which generated the highest accuracy among eight different classifiers. Finally, once the testing stage was over, we generated the list of candidates (**LoC**) for each vpin and measured the classification accuracy and average size of LoC as metrics of the effectiveness of the attack. (See end of Section

[3]Download from `http://cae.wisc.edu/~adavoodi`

4.1.2 on how LoC is generated.) We also evaluated a proximity attack in which the closest vpin was picked as the match from each vpin's LoC. All experiments ran on an Intel Core i7 - 4790 with Ubuntu 16.04 LTS and 24GB memory.

4.2.1 Comparison of Effectiveness of the Attack

Table IX shows comparison between the results from the prior work [25] and our two approaches (ML and ML-imp) for layers 4, 6, 8 as the split levels. For comparison with [25] we used the numbers reported in their paper and used an identical experimental setup. The table reports 3 metrics:

1. **—LoC—** designates the average size of the identified List of Candidates by each approach for a design;

2. **Accur.** measures the classification accuracy which is the percentage of the times that the actual match of a vpin is included in its LoC;

3. **%PA** reports the rate of successful proximity attack; the closest vpin (in terms of Hamming distance) in each LoC is picked as its match and then the percentage of correctly-guessed matches are reported.

The results are shown for three different split levels. First, **for layer 8**, both ML and ML-Imp provide similar results which are much better than [25] in all three metrics. Specifically, the |LoC| of ML and ML-Imp is about 8% of the size in [25]. Both ML and ML-Imp have similar accuracy close to 100%, compared to on-average 42.72% in [25]. In terms of the proximity attack (%PA), only the results for `superblue1` were reported in [25] so the remaining entries for this approach are left empty. The rate of successful proximity attack of ML and ML-Imp are more than 5X higher than [25], which means the LoC generated by them is not only shorter and has higher accuracy but also much more effective. Between ML and ML-Imp, we observe that ML-Imp has slightly smaller LoC size, slightly lower accuracy, and performs slightly better in proximity attack. Interestingly in `superblue10` which

	Benchmark		Original			Machine Learning			ML-Imp			Runtime	
Split		**#vpin**	**E[LS]**	**Accur.**	**% PA**	**E[LS]**	**Accur.**	**% PA**	**E[LS]**	**Accur.**	**% PA**	**ML**	**ML-Imp**
Layer 8	superblue1	7824	115.1	15.53%	1.95%	16.2	100.00%	10.58%	15.3	99.82%	11.26%	3.97 min	3.00 min
	superblue5	11018	149.4	35.63%	-	27.3	100.00%	19.51%	27.9	99.42%	21.20%	7.57 min	6.00 min
	superblue10	12888	185.4	42.45%	-	21.2	100.00%	57.49%	20.8	100.00%	60.72%	9.53 min	6.38 min
	superblue12	17312	870.4	73.13%	-	44.9	99.94%	11.72%	44.4	100.00%	11.72%	17.88 min	17.28 min
	superblue18	7518	280.7	66.88%	-	23.4	99.92%	16.68%	23.1	99.97%	18.26%	3.48 min	3.60 min
	Avg	11312	320.2	42.72%	1.95%	26.6	99.97%	23.20%	26.3	99.84%	24.63%	8.48 min	7.25 min
Layer 6	superblue1	42998	487.8	33.40%	0.76%	1712.0	83.12%	1.91%	553.4	74.65%	2.11%	11.12 hrs	4.97 hrs
	superblue5	56173	506.8	39.40%	-	1775.2	88.71%	2.07%	560.4	78.89%	2.28%	16.78 hrs	7.12 hrs
	superblue10	87212	687.9	64.03%	-	2300.8	92.65%	5.44%	740.8	82.56%	5.82%	32.48 hrs	10.20 hrs
	superblue12	75994	2527.9	73.50%	-	8383.4	97.25%	3.25%	2648.7	90.18%	3.40%	38.90 hrs	25.55 hrs
	superblue18	33596	773.6	58.43%	-	2678.7	92.78%	2.97%	793.1	83.40%	3.08%	8.97 hrs	5.30 hrs
	Avg	59194	996.8	53.75%	0.76%	3370.0	90.90%	3.13%	1059.3	81.94%	3.34%	21.65 hrs	10.73 hrs
Layer 4	superblue1	149517	885.6	58.19%	0.64%	-	-	-	912.2	75.45%	2.58%	-	1.83 hrs
	superblue5	178136	745.8	53.70%	-	-	-	-	756.4	66.14%	1.61%	-	1.90 hrs
	superblue10	215292	939.4	54.68%	-	-	-	-	877.0	64.09%	2.50%	-	1.85 hrs
	superblue12	170572	2078.8	75.67%	-	-	-	-	1868.0	82.59%	3.48%	-	3.48 hrs
	superblue18	85146	1076.9	70.13%	-	-	-	-	957.5	71.73%	2.47%	-	0.95 hrs
	Avg	159732	1145.3	62.47%	0.64%	-	-	-	1074.2	72.00%	2.53%	-	2.00 hrs

Table IX: Comparison of our two machine learning based approaches with prior work for different split levels

had one of the highest number of vpins (12888 as reported in column 3), the %PA was about 60.72% using ML-Imp indicating that most of the broken connections were correctly guessed by machine learning for this layer. **For layer 6**, compared with [25], both ML and ML-Imp have much better accuracy. However, the |LoC| in ML is more than 3.3X larger than that of [25], which was one of the motivations behind developing ML-Imp. We can see that |LoC| of ML-Imp is on-average comparable (1.06X) to [25] while it has much higher classification accuracy (on-average 81.94% compared to 53.75%). Thus, ML-Imp gives a much better overall solution. For this layer, the success rate of proximity attack remains low for all approaches indicating attacks (beyond proximity) are needed to take advantage of LoC to guess a match for each vpin.

In terms of runtime, ML-Imp is less than half of ML. While the runtimes are much higher in layer 6, they are acceptable for such large designs. (The runtime of [25] was not reported.)

4.2.2 Analysis of Feature Ranking

Based on the training samples, we measured two statistical metrics, i.e., correlation coefficient and information gain, to find the importance of each of the 9 layout features given in Section 4.1.2. Correlation coefficient is a number that quantifies correlation between two variables. Information gain is the measure of reduction in the entropy of a variable achieved by learning the state of another one.

For each feature, the metrics are calculated from the vpin pairs which were used as samples for training the ML-Imp. Specifically, one variable is the layout feature of a vpin pair, and the second variable is whether the two vpins are a match (i.e., if it is a positive or negative sample). These metrics were measured using Weka [9] and reported as a number between 0 and 1.

Figure 4.5 shows the ranking of the features for each design for layers 6 and 8. We make the following observations:

(1) The most important features are those related to the locations of the vpins and then Hamming distance between the vpins. Next, the location and Hamming distance between the pins (at the placement layers) become important. This shows the routing is more important than placement in analyzing the security of split manufacturing. Difference in the load/driver cell areas (`diffCellArea`) is the next important feature in terms of information gain. However, in one benchmark `superblue10` for layer 6, this feature is the third rank and more important than the pin-based features. **(2)** In general, for almost all the features, their correlation coefficient and information gain are decreased when going from layer 8 to layer 6. This behavior indicates that these features are not equally powerful when classifying the samples from layer 6 compared to layer 8. Thus, the classifier trained for layer 6 performed somewhat worse than layer 8, as we observed from Table IX. **(3)** Correlation coefficient and information gain of features change relative to each other with change in layer; some features that are highly dominant in layer 8 becoming less dominant when moving to layer 6. Thus, other features become relatively more important.

Going back to comparison in Table IX for layer 4, the runtime of ML was prohibitively large. For ML-Imp, the runtime was also high so we used the results of feature ranking to select a subset of highest-rank features to generate the results for ML-Imp faster. Specifically, we picked the top 7 features (all excluding `totalCellArea` and `totalWireLength`). In addition, we picked the neighborhood used in ML-Imp to include 80% positive samples as opposed to the default 90%. (Refer to Section 4.1.3 for details.) With these adjustments, the runtime of ML-Imp was only 2 hours on-average across the designs. As can be seen in Table IX, ML-Imp has similar —LoC— on layer 4 compared to [25] but with better accuracy (on-average 72% compared to 62%) and better result of proximity attack. Moreover, please note that testing and training data are not separated in [25].

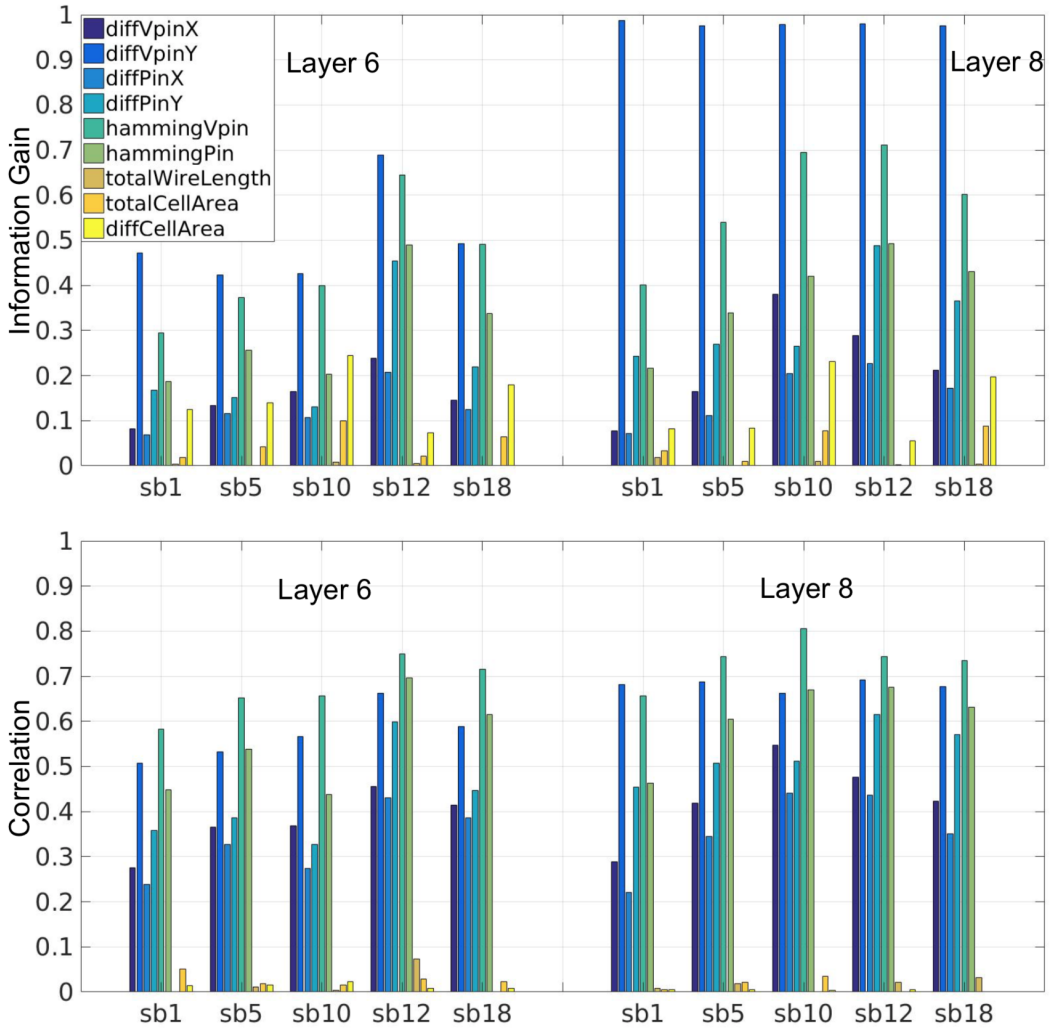

Figure 4.5: Comparison of relative ranking of the 9 layout features using Information Gain and correlation coefficient.

4.3 Conclusions

In a first of its kind approach, we developed novel and scalable machine learning techniques to analyze the security of split manufacturing on large designs. We showed significantly better results compared to those in Chapter 3, where on average for split layer 8 we increased the likelihood of our list of candidates including

the actual match from 42.72% to 99.9% while at the same time reducing the list size to be 8% of the original.

We also analyzed ranking of various layout features taken from placement, routing, and cell libraries used in the netlist, and showed the routing features are in general the most helpful to the learning process. This emphasizes the importance of using a correct setup (in terms of number of metal layers, and proper control of the routing algorithm) in generating the challenge test cases to study split manufacturing. Our studies also showed various challenges we addressed to handle the scalability issue when the split level is at the lower layers. Our machine learning scalability improvements reduced the average attack runtime on layer 6 from 21.65 hours to 10.73 hours. This further emphasizes the importance of considering computational feasibility as a constraint dictated by the experimental environment when designing an attack algorithm.

Chapter 5

A Pareto-Algebraic Routing Defense

In this chapter we analyze a custom routing algorithm as a defense against attacks by the untrusted foundry in the split manufacturing process. We propose a novel algorithm which uses a previously routed design to generate multiple alternative routing solutions. Given these multiple solutions, one will always be best in terms of wirelength overhead and one will always have the highest security for a desired allowable wirelength overhead that can be tolerated. Our algorithm uses a Pareto algebraic approach to achieve this tradeoff in the wirelength versus security space. The solutions are generated in a flexible manner controlled by the designer.

5.1 Our Procedure

This section presents our security vs. wirelength global routing procedure. We first describe how we represent candidate routes of a net for our procedure and then present the details of the procedure.

5.1.1 Modeling and Overview of Our Procedure

Our routing procedure can be described as a Pareto-algebraic approach that provides a tradeoff between security and wirelength by generating multiple global routing solutions. We use Pareto algebra because it is a mathematical framework that

provides a way to choose between options to tradeoff between metrics. This approach is novel because it allows us to generate multiple different routing solutions that offer tradeoffs between the two metrics.

In Pareto Algebra, an option that optimally trades off between metrics relative to other options is called a Pareto point. Pareto points in the context of this work are those options with higher security metric and lower wirelength that therefore dominate other options. A feasible option must generate zero overflow in the context of the broader routing solution, otherwise it is designated infeasible.

The **Security Metric (**SM**)** considered in this chapter is defined as the summation of the individual hamming distances between matching vpins at the split layer. The Hamming distance of two vpins is the distance between two vpins at the split layer measured as the sum of the horizontal x-distance plus the horizontal y-distance. A higher value of the SM metric indicates more distance on average between two matching vpins and thus higher security because it is harder for an attacker to guess the BEOL connections for matching vpins when proximity information is obfuscated in this way.

To achieve a higher security metric for a net, the net needs to be routed up through the split level. Additionally, routing a net such that the vpins are moved further apart at the split level results in a higher security metric at the cost of additional wirelength. Thus the tradeoff for a higher value of the security metric is additional wirelength.

In a description of our security metric vs. wirelength procedure, each candidate route configuration can be denoted by $(SM, WL, e_1, \ldots, e_E)$. The first term is the security metric (SM), the second term is the net wirelength (WL), and the remaining terms are the binary values to capture if the route is utilizating an edge. We use E to represent the total number of edges in the routing grid. A value of 1 for e_i indicates that the net uses that edge as part of its route, otherwise it is set

to 0. The summation of the last E entries in a configuration represents a route's wirelength and is saved as the WL entry.

A net can have multiple configurations. The j^{th} configuration of net n_i is denoted by c_{ij}. For example, in Figure 5.1, we have 2 candidate routes for net n_1 and 2 for net n_2. The 3 x 3 routing illustrations shown in Figure 5.1 are a side view where the bottom of the illustration is the lowest metal layer (layer 1), and the desired split level is represented by a dotted line. The locations of the layer 1 net pins are marked for each net. Security metric (SM) and net wirelength (WL) are given by the first two dimensions. The remaining $E = 21$ entries reflect the utilization of the routable edges; for a candidate route, there is an entry of 1 if it contains the corresponding edge.

For each net, we represent its *configuration set* as $\mathcal{C}_i = \{c_{i1}, ..., c_{i|\mathcal{C}_i|}\}$. This set designates different options to route the net. For the security vs. wirelength tradeoff problem, we designate configuration c_{ij} as a Pareto point of set $\mathcal{C}_i$ if it is at least as good as or better than other configurations in both security (higher values are better) and wirelength (lower values are better).

Pareto algebra has several operations, which we use in an iterative process to generate multiple routing solutions. These operations are *combine*, *min* (minimize), and *reduce*. We will go into the details of these operations later in this section. For additional details on computing Pareto points with Pareto algebraic operations, we refer the reader to [11, 12, 35].

Next we discuss a high level overview of our procedure to solve the security vs. wirelength tradeoff. We start by using an existing routing solution with zero overflow. These zero overflow solutions are generated with the *NCTU-GR* router [8, 22]. Before starting our procedure, we first store and fix the routes of a small portion of the nets. These nets will be excluded from consideration by our procedure. For this subset, we select nets with more than 2 (placement-level) pin connections. These nets are then designated as fixed nets whose routes will be used as part of the final

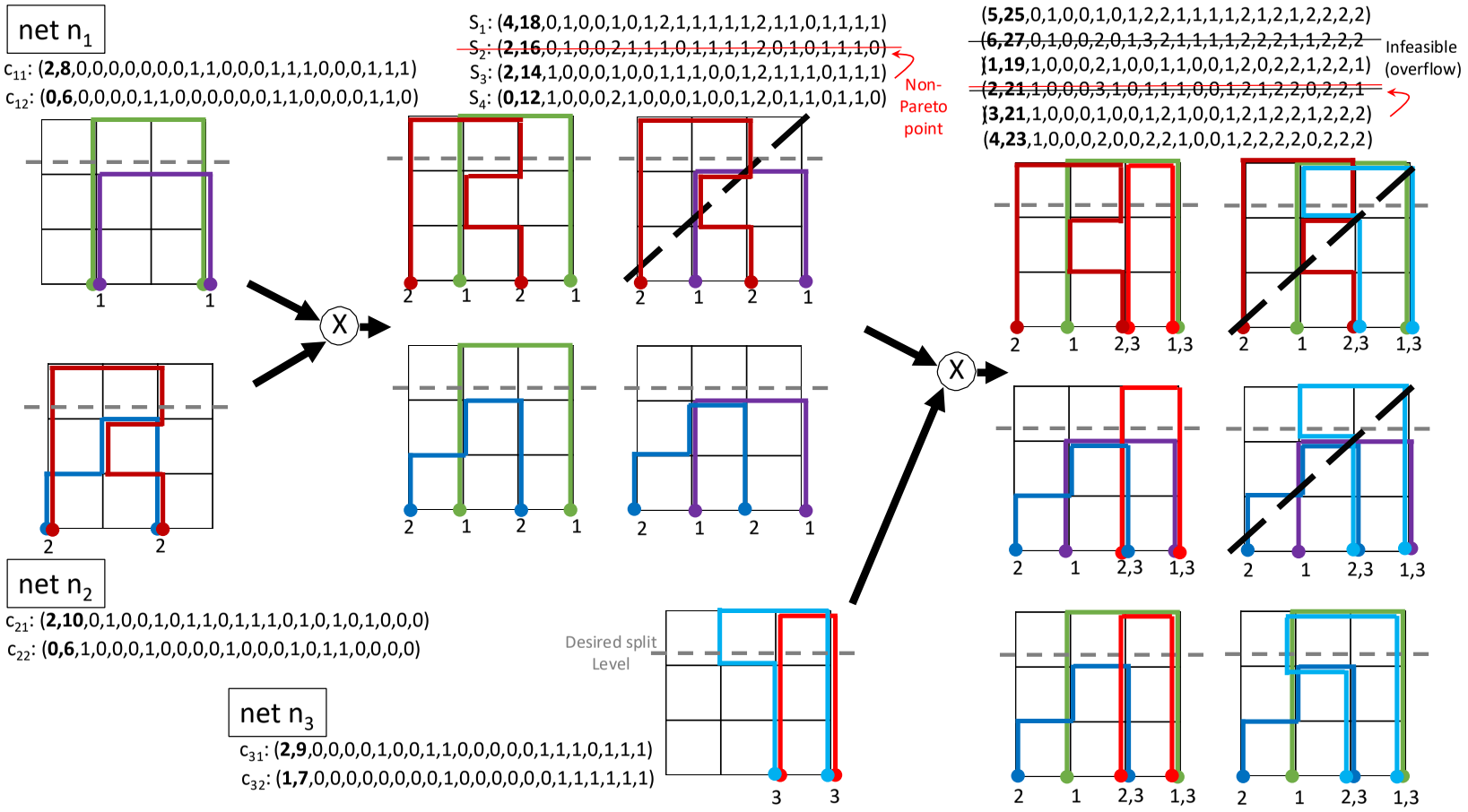

Figure 5.1: Example of the Pareto-algebraic procedure

tradeoff solution. We chose this subset of nets as fixed nets to simplify the routing functionality required to generate routing options for the remaining unfixed nets. In addition, a small number of nets with 2 pin connections are also designated as fixed nets which will be described later in this section.

The (unfixed) nets that are then considered by our algorithm are routed sequentially, in a specific order (discussed in Section 5.1.3). At each step of the algorithm for a considered (unfixed) net, we combine its various configurations with the configuration set that represents all nets that have been previously processed. While processing each step, we preserve a set of options for all nets processed so far which we refer to as a "compound configuration." In a compound configuration, each option contains a different tradeoff in the security and wirelength objectives and the E edge utilizations. The outcome of each step is a set of these compound configurations, which is denoted by $\mathcal{S} = \{s_1, s_2, ..., s_{|\mathcal{S}|}\}$, where each s_i is a compound configuration. Combining a net configuration with the compound configurations involves the Pareto-algebraic operation called *combine*, which is the same as taking the product of two sets.

After combining, we check the resulting configuration set and remove any infeasible ones (i.e., containing overflow of routing resources). We also remove those compound configurations that are not Pareto points.

Consider Figure 5.1 again, which illustrates our procedure for a representative set of 3 nets. The Figure shows a side view of the nets with various candidate routes connecting layer one pins using higher metal layer routing resources. Each edge has a capacity of 2 units. To start, net n_1 and net n_2 (each with 2 configurations) are merged into a compound configuration set containing 4 configurations.

For these four compound configurations, s_2 is superseded by s_3 and is therefore removed because it is not a Pareto point. The subsequent step merges the two configurations of n_3 with the 3 remaining compound configurations containing nets n_1 and n_2, resulting in 6 compound configurations. The final step in Figure 5.1 then

removes 2 of these compound configurations for infeasibility due to overflow (edge capacity violation). Note that one of the infeasible configurations would have also been removed because it is superseded by another configuration and therefore not a Pareto point.

Figure 5.2 illustrates the various types of routes that are presented as options for the current net in Figure 5.1. For a single net, we consider 5 options: the shortest direct routing option, routing up through the split layer, routing up through the split layer with additional space between the vpins, an A* path option, and the original *NCTU-GR* zero overflow option.

If none of the first four newly generated route options are viable during the current iteration, we mark the net as a fixed net and exclude it for consideration by our algorithm. Our procedure terminates once all unfixed nets are processed. Next we discuss this high-level procedure in more detail.

5.1.2 Details of the Procedure

Algorithm 4 gives the details of our procedure that uses a Security Wirelength Tradeoff (SWT).

The algorithm takes two input parameters: an initial configuration set $\mathcal{C}_i = \{c_{i,1}, \ldots, c_{i,|\mathcal{C}_i|}\}$ that contains routing options for net n_i in $\mathcal{N}$, as shown in Figure 5.2 and the edge capacity vector denoted by $r_{\mathcal{E}}$.

We start by initializing the set of compound configurations $\mathcal{S}$ to $\mathcal{C}_1$ for net n_1 (line 2). This results in element $s_j = c_{1,j}, \forall s_j \in \mathcal{S}$. Following this we iterate through the remaining nets using a net ordering procedure which will be described in Section 5.1.3. For each net n_i (line 3), the algorithm looks at each $s_j \in \mathcal{S}$ that contain the combinations of already-processed nets n_1 to n_{i-1} with configurations of n_i (lines 5, 6).

To determine feasibility of the combination of $c_{i,l}$ and s_j, we compute the last E dimensions by adding the elements in $c_{i,l}$ and s_j. We check these values against

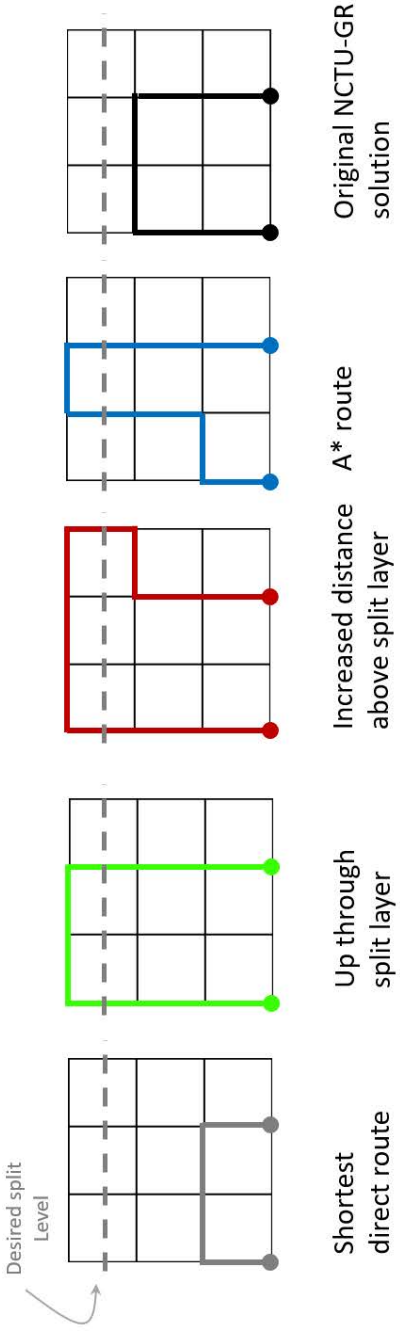

Figure 5.2: Route options for an example net

Algorithm 4 Pareto-algebraic heuristic for SWT

1: SWT($\mathcal{C}_i\ (\forall n_i \in \mathcal{N}),\ r_\mathcal{E}$)
2: $\mathcal{S} = \mathcal{C}_1$ *//so we have* $(s_1 = c_{1,1}, s_2 = c_{1,2}, \ldots, s_{|\mathcal{C}_1|} = c_{1,|\mathcal{C}_1|})$
3: **for all** net n_i, $i = 2$ to $|\mathcal{N}|$ **do** *// Section 5.1.3*
4: $\quad\mathcal{S}^{tmp} = \emptyset$
5: $\quad$**for all** $l = 1$ to $|\mathcal{C}_i|$ **do**
6: $\quad\quad$**for all** $s_j \in \mathcal{S}$ **do**
7: $\quad\quad\quad$**if** feasible$(s_j, c_{i,l}, r_\mathcal{E})$ **then**
8: $\quad\quad\quad\quad s^{tmp} = combine(s_j, c_{i,l})$ *// Fig. 5.1 and Section 5.1.1*
9: $\quad\quad\quad\quad \mathcal{S}^{tmp} = \mathcal{S}^{tmp} \cup \{s^{tmp}\}$
10: $\quad\quad\quad\quad min(\mathcal{S}^{tmp})$ *// Section 5.1.2*
11: $\quad\quad\quad$**else**
12: $\quad\quad\quad\quad fix(n_i)$ // fix net and increment priority of net n_i
13: $\quad\mathcal{S} = \mathcal{S}^{tmp}$
14: $\quad reduce(\mathcal{S}, L)$ *// Section 5.1.4*
15: $\quad sort(\mathcal{N})$ // sort nets based on updated priority *Section 5.1.3*

the edge capacity vector, $r_\mathcal{E}$ to determine if the combination would result in overflow, and if so, remove it from consideration. If the overflow feasibility check is successful, we go on to calculate the wirelength of the combination by adding the wirelength dimension from $c_{i,l}$ and s_j.

To obtain compound configuration s^{tmp} in its entirety (line 8), we also need to compute the security metric of the combination by adding up all of the hamming distances between matching vpins at the desired split level. This corresponds to adding the values of the first entry in s_j and security metric value for net n_i (given in $c_{i,l}$).

The result of the *combine* operation in line 8 is added to the set $\mathcal{S}^{tmp}$ in order to show the set of possible combinations between the configurations of n_i and the

s_js considered previously. Finally, the Pareto points are determined by applying the Pareto-algebraic min operation to $\mathcal{S}^{tmp}$. To achieve this, the latest configuration is checked to ascertain if it dominates any of the existing configurations or if it is dominated by any of them.

The algorithm then goes on to combine the configuration s_{j+1} with n_i. When every s_j is processed, set $\mathcal{S}$ is set equal to $\mathcal{S}^{tmp}$ and the process repeats with net n_{i+1}.

Since $\mathcal{S}$ can become very large, we check to see if it surpasses a user defined threshold parameter L. In Section 5.1.4, we explain how we keep a maximum of L compound configurations and eliminate the rest (line 17), providing the user a trade-off set of a maximum of L points. Our procedure stores the corresponding configuration of each net, and therefore can provide the user with the route of each net for of each of the stored L compound configurations. In the remainder of this section, we first describe our net ordering procedure and then explain the *reduce* procedure used to identify L compound configurations at each step.

5.1.3 Net Ordering

Our experience is that the quality of the end result is dependent on net ordering. When nets are well ordered, dominant solutions can be determined and infeasible ones eliminated early in the process. If nets are not well ordered, the amount of Pareto points will rapidly reach the threshold; when this happens, it becomes necessary to eliminate Pareto points through the reduce step.

We first identify congested areas in the layout and first pick those nets which fall in these areas in our ordering procedure. The congested areas are *approximated* from the location of pins for all the nets such that areas with higher pin density are considered more congested. Once congested areas are detected based on pin density, they are traversed in order of decreasing pin density. Next, the nets in each congested region are visited by our procedure. This net ordering allows for the

detection and elimination of dominated and infeasible configurations early on in Algorithm 4.

5.1.4 The Reduce Procedure

The set $\mathcal{S}$, in Algorithm 4, represents the compound configurations which were found to be Pareto points (line 16). When the size of $\mathcal{S}$ surpasses the user-defined parameter L, some of these Pareto points will be eliminated. This set of L Pareto points is selected with the objective of representing tradeoffs between our security metric and wirelength in a *uniform* manner such that the selected points are equally spaced with respect to the security metric.

Figure 5.3 depicts the projected compound configuration set $\mathcal{S}$ after processing 100K nets from benchmark `superblue1` for our security and wirelength tradeoff. To reduce the number of configurations, we identify L security metric ranges of equal size that span the difference between the maximum and minimum security metric values. For each of these ranges, we select the Pareto point that has the minimum value for wirelength and eliminate the rest.

In the example of Figure 5.3, we have set $L = 4$, so that 4 Pareto points are selected. We also show several non-Pareto points being eliminated, although this would happen before the *reduce* step. In our experiments L is set to 3. This approach results in reduced run time while still generating viable tradeoff solutions.

5.2 Experimental Results

We implemented our security vs. wirelength algorithm using a Pareto-algebraic approach as discussed in Section 5.1. Our Pareto-algebraic approach is related to the framework established in [35], which generated multiple global routing solutions that yield a tradeoff between power and wirelength by mixing routes generated for individual nets by different routers for the same design. Our approach instead starts with one global routing solution that is optimized for wirelength, and

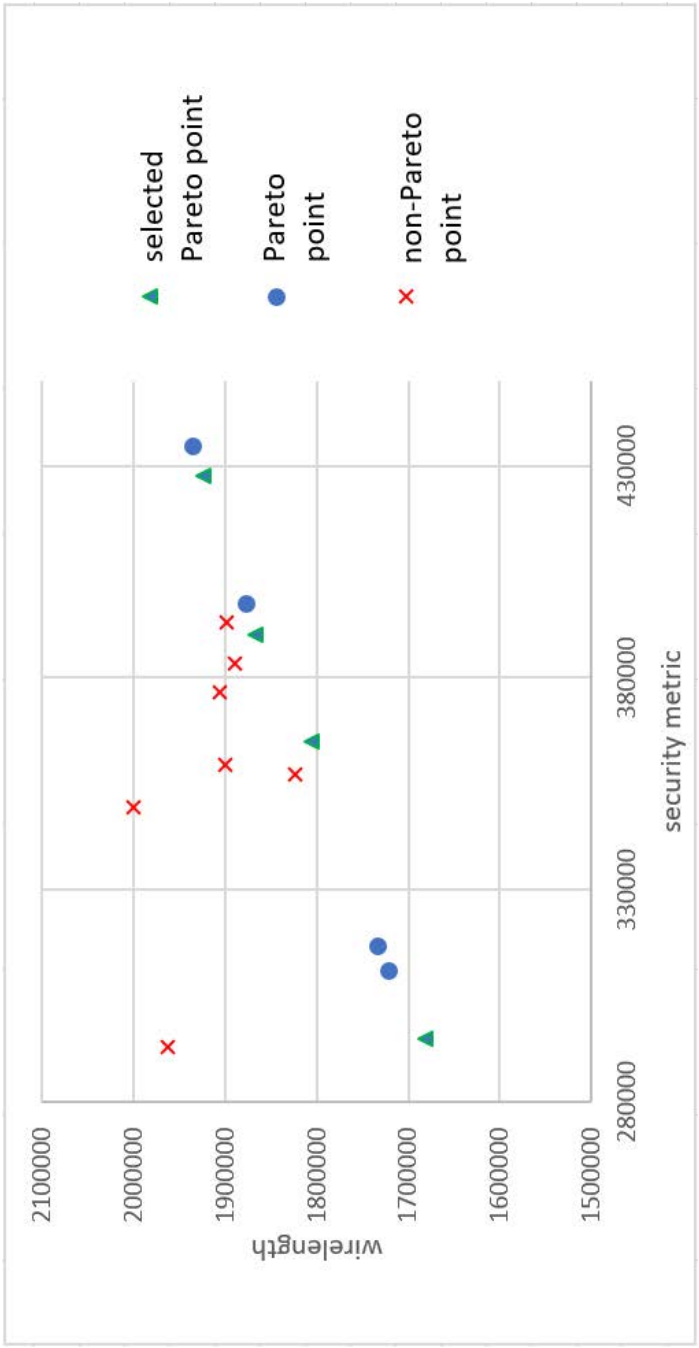

Figure 5.3: Reduce after processing 100K nets in `superblue1`.

dynamically generates additional routes per net such that in the end multiple global routing solutions are generated and yield a tradeoff between wirelength and our security metric. In the end, the user can decide which one of the generated solutions may be selected based on how much he/she is willing to trade off degradation in wirelength with improvement in the security metric. All our experiments were conducted with designs from the ISPD's 2011 benchmark suite [42] and using the same setup as in Chapters 3 and 4. The placements of these benchmarks were from the `UMich_SimPLR` tool which we obtained from the ISPD contest website. Next, an initial global routing solution was generated per benchmark using `NCTU-GR 2.0` [8, 22]; this was the wirelength-optimized solution that fed our framework. In our experiments, we considered those benchmarks for which the router was able to generate a routable (i.e., 0 overflow) solution. These included 5 of the 8 benchmarks in the suite. All solutions generated by our approach are also guaranteed not to exceed the capacities of the existing routing resources.

We implemented a variation of our framework which generated four global routing solutions (including the original wirelength-optimized solution).

5.2.1 Comparison of Security vs Wirelength Tradeoff Across Different Designs

In this experiment we set the split level to be layer 8 and study the security vs wirelength tradeoff among the four generated solutions for the 5 ISPD'11 benchmarks. We report the following metrics in Table X.

- **Total Nets:** This is the total number of nets per benchmark.
- **% Fixed Nets:** This is the number of fixed nets after running the algorithm as described in Section 5.1; recall routing options are not generated for fixed nets. Note, for solution 0, i.e., original minimum wirelength solution the percentage is always 100%.

- **Wirelength:** This is the total wirelength of all routed nets, including the fixed ones. All solutions have zero overflow.

- **Security Metric:** This is the security metric of a generated global routing solution as defined in Section 5.1; it is the summation of individual hamming distances speperating matching vpins at the split level. The higher the value of this metric, the more distance between two matching vpins and thus higher security.

- **% Increase Security Metric:** This is the percentage increase in the security metric compared to the security metric of the original, wirelength-optimized solution.

- **E[LS]:** This metric measures the number of candidate vpins in the search area of each vpin and then reports the average across all search areas. The higher value of this metric indicates a harder job for the attacker to apply various pruning strategies because there are on-average more candidates to match each vpin. The search area is defined using the routing proximity method in [25], where area is determined using the average vpin bounding box on that layer for a given benchmark. We use 'leave-one-out' cross validation to calculate the relevant routing search area. Specifically, assuming we have k benchmarks, to test each benchmark, we use the k - 1 remaining benchmarks to calculate an average search area. This cross validated search area is calculated for the given split layer. Table XI shows the average routing bounding box for each of the benchmarks per split level from the setup in [25, 50], and the resulting cross validated bounding box for split level 8.

- **% E[LS]:** This is the percentage increase in the average list size of the solution compared to the original wirelength-optimized solution.

- **FOM:** This is a Figure of Merit defined as E[LS/SA]. It is the ratio of candidate list size (LS) divided by the search area (SA), when averaged over all the search areas at a split level as described in [25]. A higher value of this metric means it is more challenging for the attacker because the density of candidates (over the same search area) will be higher.

Benchmark	Solution	Total Nets	% Fixed Nets	Wirelength	Security Metric	% Security Metric Increase	E[LS]	% E[LS] Increase	FOM E[LS/SA]
superblue1	0 (WL-optimized)	822744	100.00	14735024	668151	0.00	47.84	0.00	0.09
	1		35.38	14819009	809444	21.15	59.51	24.39	0.11
	2		35.47	15627981	1120217	67.66	77.52	62.04	0.15
	3		35.44	16021267	1280951	91.72	94.32	97.16	0.18
superblue5	0 (WL-optimized)	786999	100.00	18467069	1228984	0.00	187.62	0.00	0.26
	1		32.81	18735431	1555097	26.54	192.72	2.72	0.26
	2		32.74	19210647	1674545	36.25	210.32	12.10	0.29
	3		32.73	19225746	1687469	37.31	212.07	13.03	0.29
superblue10	0 (WL-optimized)	1085737	100.00	25093719	1735550	0.00	282.71	0.00	0.36
	1		36.41	25097485	1780416	2.59	302.79	7.10	0.39
	2		36.20	25506215	2077000	19.67	372.97	31.93	0.48
	3		36.23	25525573	2086728	20.23	381.10	34.80	0.49
superblue12	0 (WL-optimized)	1293436	100.00	22450523	1067411	0.00	790.35	0.00	0.70
	1		46.97	22925140	1123022	5.21	831.96	5.26	0.74
	2		46.87	23031677	1166467	9.28	844.34	6.83	0.75
	3		46.99	23100143	1181520	10.69	935.56	18.37	0.83
superblue18	0 (WL-optimized)	468918	100.00	11619958	660176	0.00	541.10	0.00	0.51
	1		32.93	11641806	670146	1.51	569.60	5.27	0.54
	2		32.90	11719276	703168	6.51	584.62	8.04	0.55
	3		32.96	11751597	710080	7.56	682.57	26.14	0.65

Table X: Comparison of various benchmarks for split level 8

We make the following observations from Table X.

- Solutions 0 through 3 gradually show degradation in wirelength is traded off with improvement in the security metric.
- Improvement in the security metric is also reflected in higher E[LS] and FOM; for example, the E[LS] increases from 0 to 97% in `superblue1`. Similarly, the FOM doubles in this benchmark from 0.09 in solution 0 to 0.18 in solution 3. This indicates that the security metric used within our framework has good correlation with these two metrics. Similar trend is seen in other

benchmarks, however the degree of improvement varies across the benchmarks.

- We also observe that a small increase in security metric will sometimes yield a much larger increase in candidate list size, as is the case `superblue12` between solutions 2 and 3, where 1.41 % increase in security metric yields a 11.54 % increase in average list size. This also is the case in `superblue18` between solutions 2 and 3, where 1.05 % increase in security metric yields a 18.10 % increase in average list size. These results indicate that security-wirelength algorithm will be very successful when prioritizing the higher security metric solutions for some benchmarks.

Split Level	4	6	8	
	E[BB]	E[BB]	E[BB]	cross validated
`superblue1`	77.50	146.19	207.34	89.35
`superblue5`	70.19	108.50	134.76	107.49
`superblue10`	96.04	116.54	125.85	109.72
`superblue12`	32.78	37.74	36.17	132.14
`superblue18`	43.81	58.12	60.61	126.03
overall average	64.06	93.42	112.95	

Table XI: Cross validated routing bounding box per benchmark

5.2.2 Comparison Across Split Levels

Table XII shows comparison between split levels 4, 6 and 8 for the solutions generated for the `superblue1` benchmark. As with the average candidate list size results (E[LS]) reported in Table X, the search areas used in Table XII are defined using 'leave-one-out' cross validation. We evaluate the same metrics as in the previous experiment and make the following observations.

- We observe the same trend on each layer that solutions 0 to 3 exhibit gradual improvement in the security metric in exchange for degradation in wirelength.
- Again, improvement in security metric correlates well with the E[LS] and FOM metrics. Specifically for layer 8, we observe that the percentage increase of the security metric is very closely related to the percentage increase in average candidate list size (increase in the security metric by 21.15, 67.66, and 91.72% results in an increase of average candidate list size of 24.39, 62.04, and 97.16% respectively). However, at the lower layers, 6 and 4, an increase in security metric results in larger increase in candidate list size. From these results we can observe that there is a positive correlation between these two metrics, but also that the relationship varies by split level.
- The percentage of fixed nets remains fairly constant for solutions 0 to 3. This result indicates that the algorithm is successful in using a wirelength-security tradeoff to re-route roughly two-thirds of the nets, independent of the target split level.

Split level	Solution	Total Nets	% Fixed Nets	Wirelength	Security Metric	% Security Metric Increase	E[LS]	% E[LS] Increase	FOM E[LS/SA]
8	WL-optimized	822744	100.00	14735024	668151	0.00	47.84	0.00	0.09
	1		35.38	14819009	809444	21.15	59.51	24.39	0.11
	2		35.47	15627981	1120217	67.66	77.52	62.04	0.15
	3		35.44	16021267	1280951	91.72	94.32	97.16	0.18
6	WL-optimized	822744	100.00	14735024	2889504	0.00	197.21	0.00	0.47
	1		35.37	14845400	2893001	0.12	243.94	23.70	0.58
	2		35.54	16009829	3473135	20.20	272.22	38.04	0.65
	3		35.46	16882663	3947411	36.61	287.99	46.03	0.69
4	WL-optimized	822744	100.00	14735024	5509221	0.00	404.46	0.00	1.69
	1		35.91	16312215	5992308	8.77	584.43	44.50	2.44
	2		35.97	16647098	6179635	12.17	591.58	46.26	2.46
	3		36.21	19142044	7334781	33.14	893.61	120.94	3.72

Table XII: Comparison of benchmark `superblue1` for various split levels

Figure 5.4 shows the security vs. wirelength tradeoff curves for layers 4, 6 and 8 for solutions of `superblue1` shown in Table X. The red points indicate the position of the original minimum wirelength solution on each curve, while the remaining blue dots are solutions 1 to 3 generated by our framework using solution 0. The three curves exhibit similar slopes, defined here as change in security metric over change in wirelength (0.44 for level 8, 0.50 for layer 6, and 0.42 for layer 4). This is similar to the overall slope of 0.46 across the 5 benchmarks in Table XII.

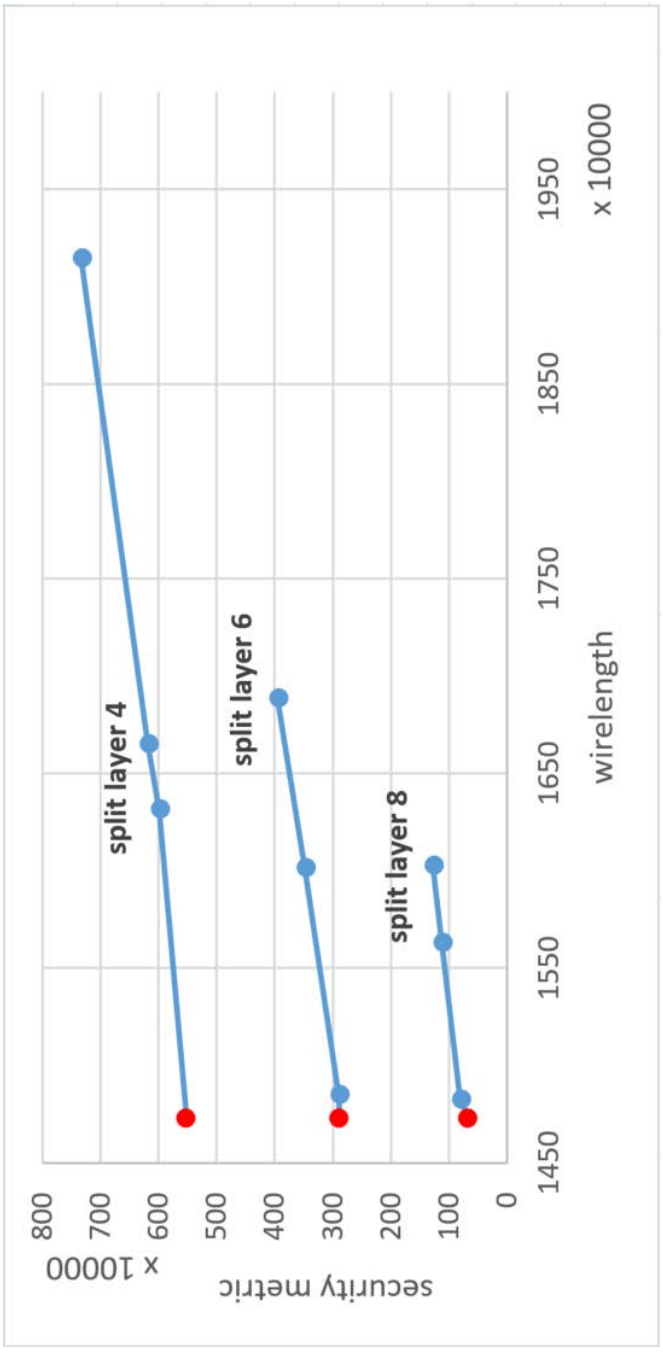

Figure 5.4: Security vs. wirelength tradeoff for layers 4, 6 and 8 in `superblue1`

5.3 Conclusions

Using a Pareto algebraic framework, we designed a new algorithm that optimizes the tradeoff between additional wirelength for additional security in the split manufacturing process. Our algorithm takes an already routed design as input and from that we generate multiple solutions, where one solution has the best wirelength overhead and one solution has the highest security for an amount of wirelength overhead that can be tolerated. A designer also has control over the target split level considered by our algorithm, such that the solutions optimized for that split level. The algorithm provides a defense that can protect a design against reverse engineering by an untrusted foundry.

In our experiments we show that our algorithm produces a measurable improvement in security in exchange for increased wirelength. On average, per unit increase of wirelength, our security metric increases by 0.46. As our security metric increases we also observe a percentage increase in the size of the of candidates that a proximity attack identifies for each vpin. When security is increased by 21.15, 67.66, and 91.72%, for example, the average candidate list size increases by 24.39, 62.04, and 97.16% respectively.

Chapter 6

Conclusions

The ever present and increasing need for security in the fabrication of IC using split manufacturing drives studies in how to achieve this security. In this dissertation, the issue of security has been carefully studied from the point of view of an untrusted foundry trying to reverse engineer a design using a proximity-based attack based on the partial information the foundry is given.

First in Chapter 3 we explored several techniques using both placement and routing information to define proximity. Each technique used intuitive ways to define the search neighborhood around each vpin. In addition, we proposed a blockage insertion technique at the split level, showing that this technique decreases the effectiveness of proximity attacks.

Our experiments in Chapter 3 led us to the conclusion that proximity information alone is not sufficient to allow an attacker to guess the connections specified by the BEOL in order to reverse engineer the overall circuit design. However, we demonstrated that our proximity techniques (especially `crouting`) can often be effective in narrowing down the list of candidates significantly, while keeping the actual matching vpin in the list. We have shown that it is important to define the search area as a function of routing congestion, given that `crouting` was the most effective proximity technique.

Next, the role of machine learning in refining a defense against these proximity attacks is a significant contribution in this endeavor, as discussed in Chapter 4. We

found machine learning improved the effectiveness of the proximity attack developed in Chapter 3 by more than doubling the likelihood of our list of candidates including the actual match, while simultaneously reducing the size of the list by more than a factor of ten.

Additionally, our studies also addressed the scalability issue encountered when a design is split at lower levels. Our scalability improvements cut the average attack runtime in half on layer 6. This emphasized computational feasibility can be an important constraint to consider when designing an attack algorithm.

Finally, in Chapter 5 we used a Pareto algebraic framework to implement a novel algorithm to help defend against reverse engineering attacks by an untrusted foundry. Based on design routed beforehand and a target split layer, our algorithm generates multiple new solutions. Of these solutions, one has the best security for an amount of wirelength overhead that can be tolerated, and one has the least amount of wirelength overhead.

We show that per one unit of additional wirelength, our algorithm proposed in Chapter 5 increases our security metric by 0.46. We also observe a roughly equal percentage increase in candidate list size, making an attack more difficult, for a corresponding increase in security.

LIST OF REFERENCES

[1] Y. Alkabani. Hardware security and split fabrication. In *Design & Test Symp. (IDT)*, pages 59–64. IEEE, 2016.

[2] C. J. Alpert and G. E. Tellez. The importance of routing congestion analysis. In *Design Automation Conf.*, DAC.com Knowledge Center Online Article, 2010.

[3] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan. Hardware Trojan attacks: threat analysis and countermeasures. *Proc. of the IEEE*, 102(8):1229–1247, 2014.

[4] S. Chen and R. Vemuri. Improving the security of split manufacturing using a novel BEOL signal selection method. In *Great Lakes Symp. on VLSI*, pages 135–140. ACM, 2018.

[5] Z. Chen, P. Zhou, T. Ho, and Y. Jin. How secure is split manufacturing in preventing hardware Trojan? In *Hardware-Oriented Security and Trust (Asian-HOST)*, pages 1–6. IEEE, 2016.

[6] C. Clavier, Q. Isorez, D. Marion, and A. Wurcker. Complete reverse-engineering of AES-like block ciphers by SCARE and FIRE attacks. *Cryptography and Communications*, 7(1):121–162, Springer, 2015.

[7] F. Corno, M. S. Reorda, and G. Squillero. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Design & Test of Computers*, 17(3):44–53, 2000.

[8] K.-R. Dai, W.-H. Liu, and Y.-L. Li. NCTU-GR: efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 20(3):459–472, 2012.

[9] F. Eibe, M. A. Hall, and I. H. Witten. The WEKA workbench. online appendix for "data mining: practical machine learning tools and techniques". *Morgan Kaufmann, Fourth Edition*, 2016.

[10] L. Feng, Y. Wang, J. Hu, W. Mak, and J. Rajendran. Making split fabrication synergistically secure and manufacturable. In *International Conf. on Computer-Aided Design*, pages 313–320. IEEE, 2017.

[11] M. Geilen and T. Basten. A calculator for Pareto points. In *Design, Automation, and Test in Europe (DATE)*, pages 16–20. IEEE, 2007.

[12] M. Geilen, T. Basten, B. D. Theelen, and R. Otten. An algebra of Pareto points. *Fundamenta Informaticae*, 78(1):35–74, IOS Press, 2007.

[13] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris. Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain. *Proc. of the IEEE*, 102(8):1207–1228, 2014.

[14] M. C. Hansen, H. Yalcin, and J. P. Hayes. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test of Computers*, 16(3):72–80, 1999.

[15] B. Hill, R. Karmazin, C. T. O. Otero, J. Tse, and R. Manohar. A split-foundry asynchronous FPGA. In *Custom Integrated Circuits Conf.*, pages 1–4. IEEE, 2013.

[16] C.-H. Hsu, H.-Y. Chen, and Y.-W. Chang. Multilayer global routing with via and wire capacity considerations. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 29(5):685–696, 2010.

[17] F. Imeson, A. Emtenan, S. Garg, and M. V. Tripunitara. Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation. In *USENIX Conf. on Security*, pages 495–510, 2013.

[18] M. Jagasivamani, P. Gadfort, M. Sika, M. Bajura, and M. Fritze. Split-fabrication obfuscation: metrics and techniques. In *International Symp. on Hardware-Oriented Security and Trust*, pages 7–12. IEEE, 2014.

[19] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas. Handling imbalanced datasets: a review. *GESTS International Trans. on Computer Science and Engineering*, 30(1):25–36, 2006.

[20] M. Li, B. Yu, Y. Lin, X. Xu, W. Li, and D. Pan. A practical split manufacturing framework for Trojan prevention via simultaneous wire lifting and cell insertion. In *Asia and South Pacific Design Automation Conf.*, pages 265–270. IEEE, 2018.

[21] D. Liu, B. Yu, S. Chowdhury, and D. Z. Pan. Incremental layer assignment for critical path timing. In *Design Automation Conf.*, pages 1–6. ACM, 2016.

[22] W. Liu, W. Kao, Y. Li, and K. Chao. NCTU-GR 2.0: Multithreaded collision-aware global routing with bounded-length maze routing. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 32(5):709–722, 2013.

[23] S. Madani and M. Bayoumi. A security-aware pre-partitioning technique for 3D integrated circuits. In *International Workshop on Microprocessor and SOC Test and Verification (MTV)*, pages 57–61. IEEE, 2017.

[24] J. Magaña, D. Shi, and A. Davoodi. Are proximity attacks a threat to the security of split manufacturing of integrated circuits? In *International Conf. on Computer-Aided Design*, pages 1–7. IEEE, 2016.

[25] J. Magaña, D. Shi, J. Melchert, and A. Davoodi. Are proximity attacks a threat to the security of split manufacturing of integrated circuits? *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 25(12):3406–3419, 2017.

[26] C. T. O. Otero, J. Tse, R. Karmazin, B. Hill, and R. Manohar. Automatic obfuscated cell layout for trusted split-foundry design. In *International Symp. on Hardware-Oriented Security and Trust*, pages 56–61. IEEE, 2015.

[27] S. Patnaik, M. Ashraf, J. Knechtel, and O. Sinanoglu. Raise your game for split manufacturing: restoring the true functionality through BEOL. In *Design Automation Conf.*, page 140. ACM, 2018.

[28] S. Patnaik, J. Knechtel, M. Ashraf, and O. Sinanoglu. Concerted wire lifting: enabling secure and cost-effective split manufacturing. In *Asia and South Pacific Design Automation Conf.*, pages 251–258. IEEE, 2018.

[29] J. Rajendran, O. Sinanoglu, and R. Karri. Is split manufacturing secure? In *Design, Automation and Test in Europe (DATE)*, pages 1259–1264. IEEE, 2013.

[30] A. Senguptay, S. Patnaik, J. Knechtelz, M. Ashraf, S. Garg, and O. Sinanoglu. Rethinking split manufacturing: an information-theoretic approach with secure layout techniques. In *International Conf. on Computer-Aided Design*, pages 329–326. IEEE, 2017.

[31] D. Shi and A. Davoodi. TraPL: track planning of local congestion for global routing. In *Design Automation Conf.*, page 19. ACM, 2017.

[32] D. Shi, E. Tashjian, and A. Davoodi. Dynamic planning of local congestion from varying-size vias for global routing layer assignment. In *Asia and South Pacific Design Automation Conf.*, pages 372–377. IEEE, 2016.

[33] D. Shi, E. Tashjian, and A. Davoodi. Dynamic planning of local congestion from varying-size vias for global routing layer assignment. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 36(8):1301–1312, 2017.

[34] Q. Shi, K. Xiao, D. Forte, and M. Tehranipoor. Securing split manufactured ICs with wire lifting obfuscated built-in self-authentication. In *Great Lakes Symp. on VLSI*, pages 339–344. ACM, 2017.

[35] H. Shojaei, A. Davoodi, and T. Basten. Collaborative multiobjective global routing. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 21(7):1308–1321, 2013.

[36] H. Shojaei, A. Davoodi, and J. T. Linderoth. Congestion analysis for global routing via integer programming. In *International Conf. on Computer-Aided Design*, pages 256–262. IEEE, 2011.

[37] M. Tehranipoor and F. Koushanfar. A survey of hardware Trojan taxonomy and detection. *IEEE Design & Test of Computers*, 27(1):10–25, 2010.

[38] R. Torrance and D. James. The state-of-the-art in semiconductor reverse engineering. In *Design Automation Conf.*, pages 333–338. ACM, 2011.

[39] K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi. Building trusted ICs using split fabrication. In *International Symp. on Hardware-Oriented Security and Trust*, pages 1–6. IEEE, 2014.

[40] K. Vaidyanathan, R. Liu, E. Sumbul, Q. Zhu, F. Franchetti, and L. Pileggi. Efficient and secure intellectual property (IP) design with split fabrication. In *International Symp. on Hardware-Oriented Security and Trust*, pages 13–18. IEEE, 2014.

[41] J. Valamehr, T. Sherwood, R. Kastner, D. Marangoni-Simonsen, T. Huffmire, C. Irvine, and T. Levin. A 3-D split manufacturing approach to trustworthy system development. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 32(4):611–615, 2013.

[42] N. Viswanathan, C. J. Alpert, C. C. N. Sze, Z. Li, G.-J. Nam, and J. A. Roy. The ISPD-2011 routability-driven placement contest and benchmark suite. In *International Symp. on Physical Design*, pages 141–146. ACM, 2011.

[43] Y. Wang, T. Cao, J. Hu, and J. Rajendran. Front-end-of-line attacks in split manufacturing. In *International Conf. on Computer-Aided Design*, pages 321–328. IEEE, 2017.

[44] Y. Wang, P. Chen, J. Hu, G. Li, and J. Rajendran. The cat and mouse in split manufacturing. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 26(5):805–817, 2018.

[45] Y. Wang, P. Chen, J. Hu, and J. Rajendran. The cat and mouse in split manufacturing. In *Design Automation Conf.*, pages 1–6. ACM, 2016.

[46] Y. Wang, P. Chen, J. Hu, and J. Rajendran. Routing perturbation for enhanced security in split manufacturing. In *Asia and South Pacific Design Automation Conf.*, pages 605–510. IEEE, 2017.

[47] K. Xiao, D. Forte, and M. M. Tehranipoor. Efficient and secure split manufacturing via obfuscated built-in self-authentication. In *International Symp. on Hardware-Oriented Security and Trust*, pages 14–19. IEEE, 2015.

[48] Y. Xie, C. Bao, and A. Srivastava. Security-aware design flow for 2.5D IC technology. In *International Workshop on Trustworthy Embedded Devices*, pages 31–38. ACM, 2015.

[49] P.-L. Yang and M. Marek-Sadowska. Making split-fabrication more secure. In *International Conf. on Computer-Aided Design*, pages 1–8. IEEE, 2016.

[50] B. Zhang, J. Magaña, and A. Davoodi. Analysis of security of split manufacturing using machine learning. In *Design Automation Conf.*, page 141. ACM, 2018.

[51] X. Zhang and M. Tehranipoor. Case study: Detecting hardware Trojans in third-party digital IP cores. In *International Symp. on Hardware-Oriented Security and Trust*, pages 67–70. IEEE, 2011.